

I. Introduction	2
a. X3D	2
b. Tutorial -- Generating Initial Grids Using the X3D Command Language:	3
1. <u>Define mesh objects</u>	3
2. <u>Define an enclosing volume</u>	4
3. <u>Define interior interfaces</u>	5
4. <u>Divide the enclosing volumes into regions</u>	7
5. <u>Assign material types to the regions</u>	8
6. <u>Distribute points within the volume</u>	9
7. <u>Connect the points into tetrahedra</u>	12
II. Mesh Objects	15
a. Mesh Object Definition	15
b. Command Interface	18
c. FORTRAN Interface	18
d. Mesh Object Connectivity	20
III. X3D Commands:	24
a. Conventions	24
b. Alphabetic Listing of X3D Commands	26
<u>ADDMESH</u>	26
<u>ASSIGN</u>	26
<u>CMO</u>	26
<u>COPYPTS</u>	34
<u>COORDSYS</u>	35
<u>DOPING</u>	36
<u>DUMP</u>	37
<u>EDIT</u>	37
<u>ELTSET</u>	38
<u>EXTRACT</u>	38
<u>FIELD</u>	39
<u>FILTER</u>	41
<u>FINISH</u>	41
<u>GENIEE</u>	41
<u>HELP</u>	41
<u>HEXTOTET</u>	42
<u>INFILE</u>	42
<u>INPUT</u>	42
<u>INTERSECT</u>	42
<u>LOG</u>	43
<u>MERGE</u>	43
<u>MREGION</u>	43
<u>OFFSETSUF</u>	44
<u>PSTATUS</u>	45
<u>PSET</u>	45
<u>QUADXY</u>	46
<u>QUADXYZ</u>	46
<u>READ</u>	47
<u>RECON</u>	47
<u>REFINE</u>	47

<u>REGION</u>	56
<u>REGNPTS</u>	56
<u>RESETPTS</u>	57
<u>RM</u>	58
<u>RMMAT</u>	59
<u>RMPOINT</u>	59
<u>RMREGION</u>	59
<u>RMSPHERE</u>	59
<u>RMSURF</u>	60
<u>ROTATELN</u>	60
<u>ROTATEPT</u>	60
<u>RZ</u>	61
<u>RZBRICK</u>	62
<u>RZS</u>	62
<u>SCALE</u>	63
<u>SEARCH</u>	64
<u>SETPTS</u>	64
<u>SETTETS</u>	64
<u>SMOOTH</u>	65
<u>SURFACE</u>	67
<u>SURFPTS</u>	70
<u>TRANS</u>	71
<u>ZQ</u>	71
IV. Interfacing User Routines to X3D	73
a. Building an executable and running X3D.	73
b. Issuing Commands from a user program.	74
c. Writing user commands	75
X3D REFERENCES:	86

I. Introduction

a. X3D

X3D is a library of user callable tools that provide mesh generation, mesh optimization and dynamic mesh maintenance in three dimensions for a variety of applications. Geometric regions within arbitrarily complicated geometries are defined as combinations of bounding surfaces, where the surfaces are described analytically or as collections of points in space. A variety of techniques for distributing points within these geometric regions are provided. Mesh generation uses a Delaunay tetrahedralization algorithm that respects material interfaces and assures that there are no negative coupling coefficients. The data structures created to implement this algorithm are compact and powerful and expandable to include hybrid meshes as well as tetrahedral meshes.

Mesh refinement and smoothing are available to modify the mesh to provide more resolution in areas of interest. Mesh refinement adds nodes to the mesh based on geometric criteria such as edge length or based on field variable criteria such change in field. Mesh smoothing moves nodes to adapt the mesh to field variable measures, and, at the same time, maintains quality element shape. Mesh elements may become distorted as mesh nodes move during a time dependent simulation or are added as a result of refinement operations. Mesh reconnection via a series of edge flips will maintain the non-negative coupling coefficient criterion of the mesh while eliminating highly distorted elements.

An additional requirement of time dependent simulations is that as interface surfaces move, the corresponding region definitions must respond dynamically. As surfaces collide, the mesh must respond by merging points and effectively squeezing out the material between the colliding surfaces. X3D provides the necessary tools for time dependent simulations.

b. Tutorial -- Generating Initial Grids Using the X3D Command Language:

The steps involved in generating three dimensional grids in the X3D command language are:

1. Define mesh objects.
2. Define an enclosing volume.
3. Define interior interfaces.
4. Divide the enclosing volume into regions.
5. Assign material types to the regions.
6. Distribute points within the volume.
7. Connect the points into tetrahedra

Detailed descriptions of the X3D commands are given in Section III. This tutorial covers just the commands needed to generate a simple grid. The tutorial will explain how to generate a grid in a unit cube containing two materials separated by a plane.

1. Define mesh objects

Define all Mesh Objects to be used in this problem using the **cmo/create** command. The **cmo/create** command establishes an empty Mesh Object data structure (see Section II.a for a description). For this example we will need only a single 3D Mesh Object:

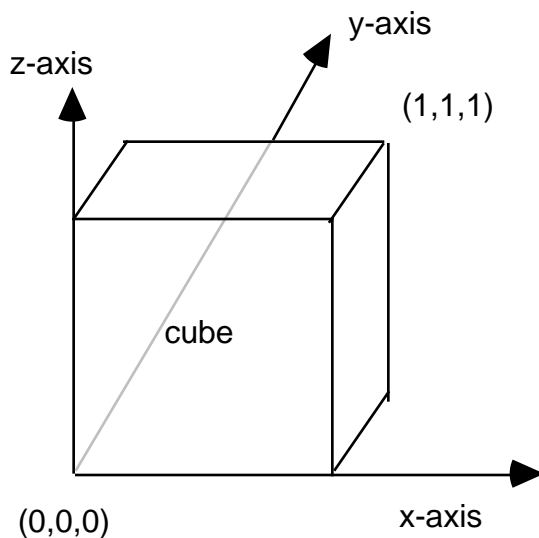
* create a 3D tetrahedral mesh object and name it *3dmesh*
cmo/create/3dmesh/

2. Define an enclosing volume

Define an enclosing volume using the **surface** command. Since we are defining an exterior boundary, the boundary type is **reflect**. The next item of information needed is the geometry of the volume; some common geometry types are **box**, **cylinder**, **sphere**. Geometry types, **box** and **sphere**, define closed volumes; whereas a **cylinder** is open on both ends and must be capped by planes. Along with the geometry type, the extent of the volume is defined by specifying for the box its corners, or for the cylinder its radius and end point of its axis of rotation. The enclosing volume must be convex. Complicated enclosing volumes can be described by their bounding surfaces including planes and sheets. Some simple examples of enclosing volumes are:

* unit cube

surface/cube/reflect/box/0.0,0.0,0.0/1.0,1.0,1.0/

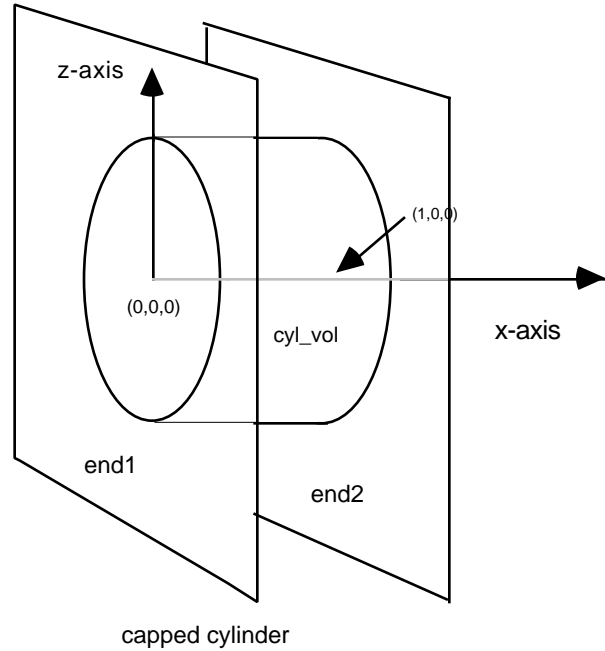
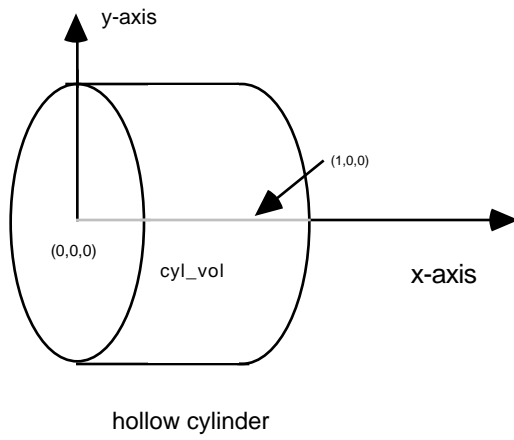


* cylinder whose axis is the x axis with radius 1 and height 1

surface/cyl_vol/reflect/cylinder/0.,0.,0./1.,0.,0./1./

surface/end1/reflect/plane/0.,0.,0./0.,0.,1./0.,1.,1./

surface/end2/reflect/plane/1.,0.,0./1.,0.,1./1.,1.,1./

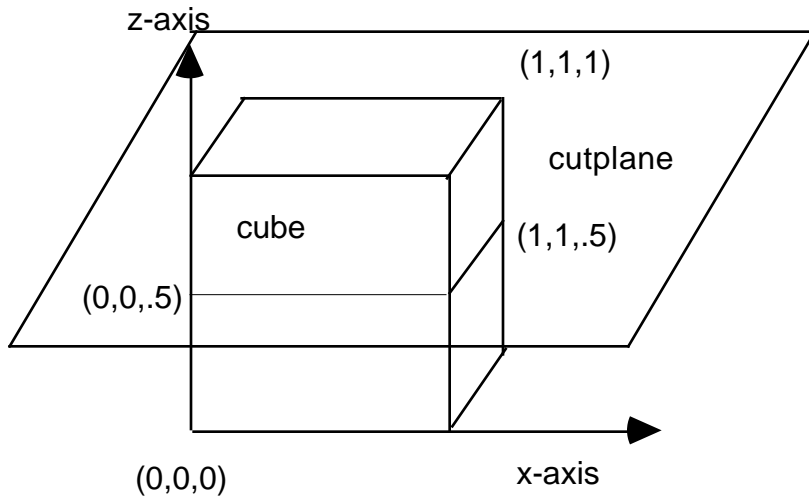


3. Define interior interfaces

Interfaces are defined with the **surface** command. In this case the boundary type is **interface**. If the command defines a volume (e.g. box, cylinder) then the interface is the surface of the volume defined. If the command defines a plane or sheet then the interface is the plane or sheet. It is important to remember that planes are infinite and that the order of points specifying the plane determines a normal to the plane in the usual right-hand-rule sense (see Section III.a.9). This direction is important in determining regions. In order to divide the unit cube defined above in half vertically, define a plane by:

surface/cutplane/interface/plane/0.,0.,.5/1.,0.,.5/1.,1.,.5/

The normal to this plane points in the positive z direction.

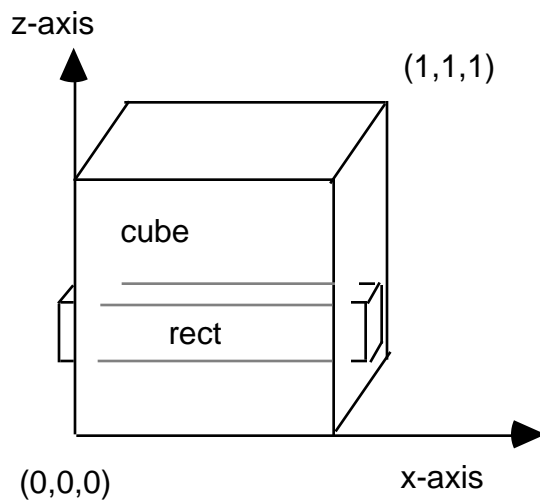


'cube' bisected by the infinite plane 'cutplane'

Interfaces must not be coincident with reflective boundaries. For example to embed a rectangle inside a cube, it is necessary to extend the ends of the rectangle beyond the cube to avoid coincident reflective and interface surfaces:

surface/cube/reflect/box/0.0,0.0,0.0/1.0,1.0,1.0/

surface/rect/interface/box/-0.1,0.5,0.2/1.1,0.6,0.5/



'cube' with embedded 'rect', 'rect' extended beyond planar surfaces of 'cube' to avoid coincident interface and reflective surfaces

4. Divide the enclosing volumes into regions

The **region** command is used to divide the enclosing volume into regions. The operators **lt**, **le**, **gt**, and **ge** are applied to previously defined surfaces according to the following rules.

lt -- if the surface following is a volume then **lt** means inside not including the surface of the volume. If the surface is a plane or a sheet **lt** means the space on the side of the plane or sheet opposite to the normal not including the plane or sheet itself.

le -- if the surface following is a volume then **le** means inside including the surface of the volume. If the surface is a plane or a sheet **le** means the space on the side of the plane or sheet opposite to the normal including the plane or sheet itself.

gt -- if the surface following is a volume then **gt** means outside not including the surface of the volume. If the surface is a plane or a sheet **gt** means the space on the same side of the plane or sheet as the normal not including the plane or sheet itself.

ge -- if the surface following is a volume then **ge** means outside including the surface of the volume. If the surface is a plane or a sheet **ge** means the space on the same side of the plane or sheet as the normal including the plane or sheet itself.

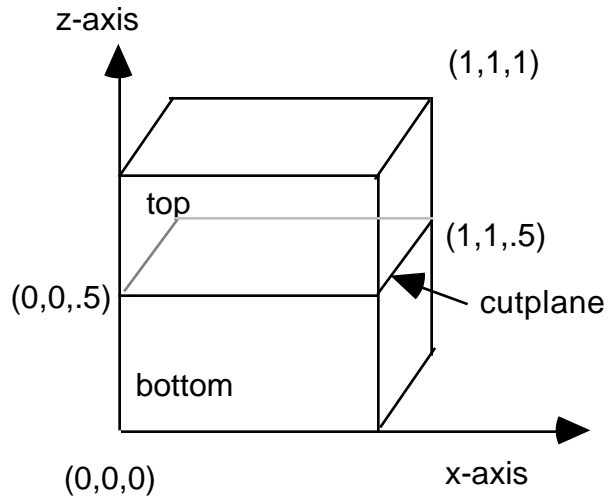
The operators **or**, **and**, and **not** applied surfaces mean union, intersection and complement respectively. The operators **or**, **and**, and **not** applied to relational operators are the normal logical operators. Parentheses are used for nesting. Spaces are required as delimiters to separate operators and operands. To define the two regions created by the plane bisecting the unit cube:

region/top/ le cube and gt cutplane /

region/bottom/ le cube and le cutplane /

The region *bottom* contains the interface *cutplane*; *top* contains none of the interface. Interior interfaces must be included in one and only one region.

If a region touches an external boundary, include the enclosing volume in **region** and **mregion** commands. For example, the regions *top* and *bottom* are enclosed in the volume *cube*

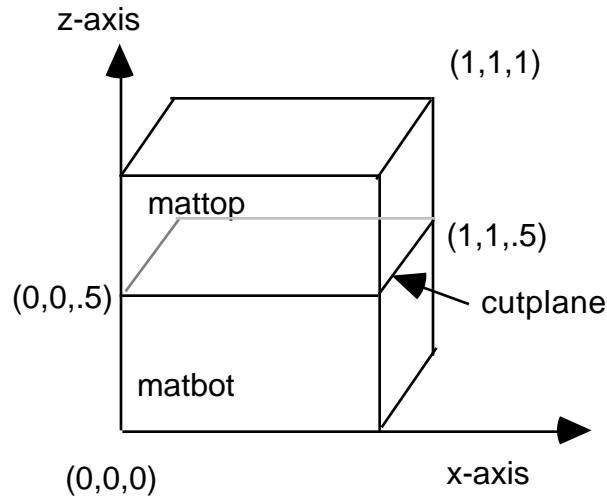


'cube' consisting of two geometric regions: 'top' and 'bottom'

5. Assign material types to the regions

Assign materials to regions using the **mregion** command. This command has similar syntax to the **region** command except that the interface should not be assigned to any material region. To assign two materials, *mattop* and *matbot*, to the regions *top* and *bottom*:

```
mregion/mattop/ le cube and gt cutplane /
mregion/matbot/ le cube and lt cutplane /
```

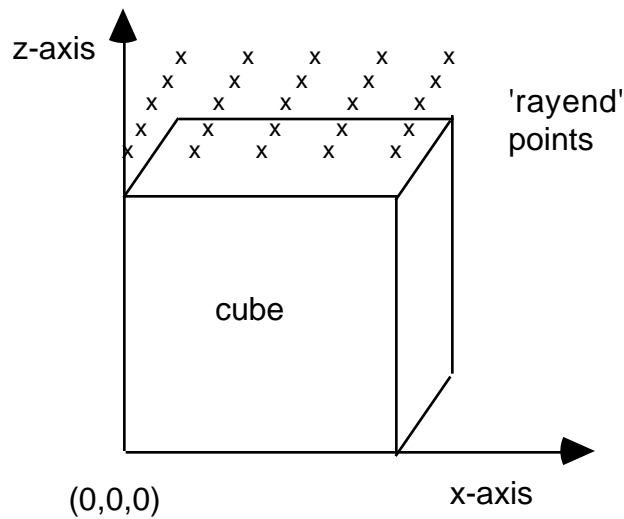


'cube' containing two materials: 'mattop' and 'matbot'

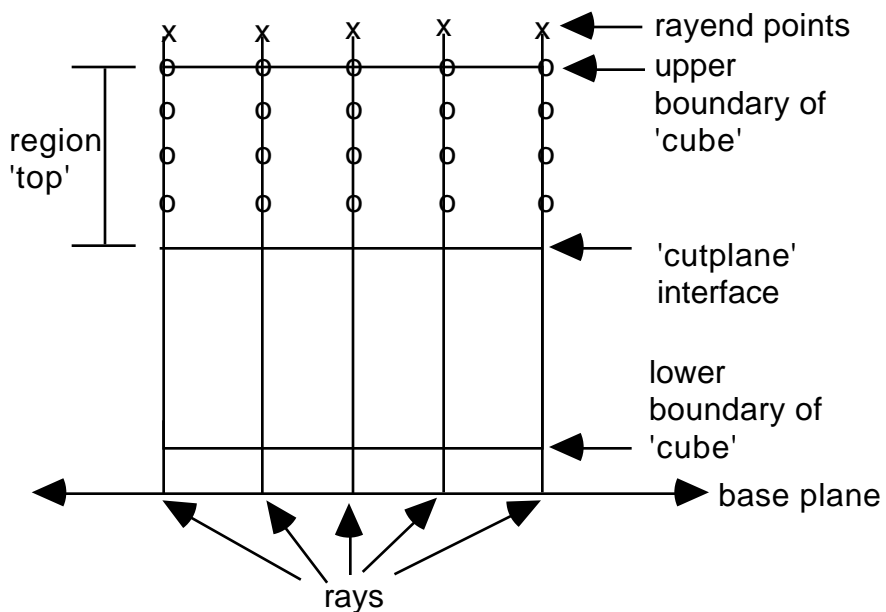
6. Distribute points within the volume

Points are distributed within regions using Cartesian, cylindrical or spherical coordinates by constructing rays that travel through regions and distributing points along these rays. For this example, points are distributed using Cartesian coordinates. The rays are specified by defining a set of points and a plane. For each point in the set, a ray is constructed normal to the plane passing through the point. In general rays are constructed in sets, each set is specified by a single plane and a set of points. The **rz** command is used to create the points. The **regnpts** command is used to specify the plane, to specify the region, and to specify the number of points to be distributed along the rays. The points and the plane should lie outside the enclosing volume and on opposite sides. The normal to the plane should point toward the point. As rays are created, if they do not pass through the specified region, no points are distributed. Points may be spaced evenly along the ray or they may be spaced according to a ratio. The following commands will place points in the unit cube.

```
* create 25 points (5x5x1) in a plane above the unit cube
* place points on the boundaries in the x and y directions (1,1,0)
rz/xyz/5,5,1/0.,0.,1.1/1.,1.,1.1/1,1,0/
* give the points defined by the rz command the name, rayend
pset/rayend/seq/1,0,0/
```



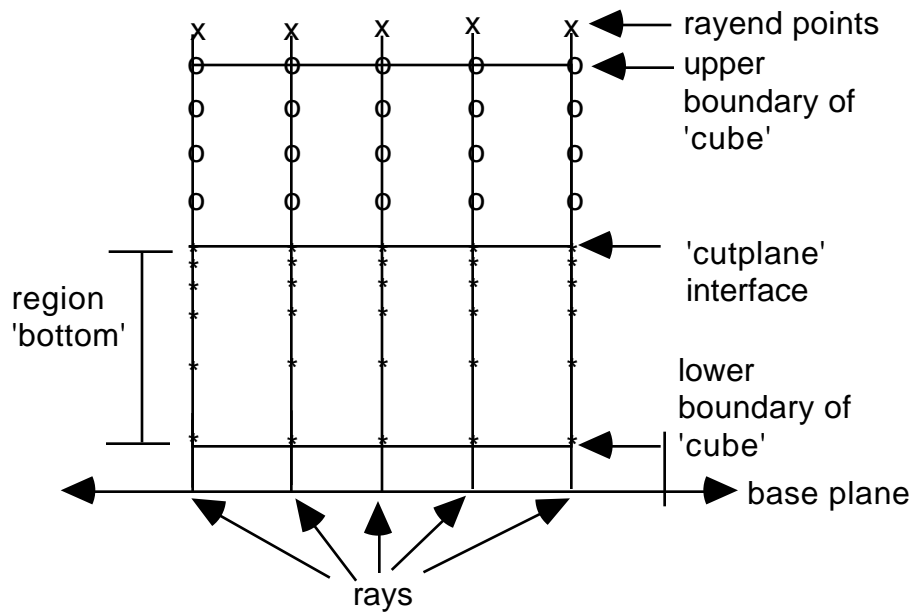
- * create rays between points in *rayend* and the plane below the cube
 - * distribute 3 points along these rays in the region *top*
 - * add one point at the upper external boundary for each ray
 - * will get 4 points total along each ray in region *top*
 - * "**pset,get,rayend**" refers to all the points named *rayend*
 - * the three points: (0.,0.,-1), (0.,1.,-1), (1.,1.,-1)
 - * define a plane whose normal points toward the *rayend* points
- regnpts/top/3/pset,get,rayend/xyz/0.,0.,-1/0.,1.,-1/1.,1.,-1/0,0/**



front face of cube showing one row of 'rayend' points and one set of 5 rays. Points are distributed in the region 'top'.

- * distribute 4 points along these rays in the region *bottom*
- * add one point at the lower external boundary for each ray
- * add one point at the material interface for each ray since
- * *bottom* contains the interface - a total of 5 points for each ray.
- * points will be distributed such that the ratio of distances between
- * any two consecutive pairs of points is 0.6 traveling from the source
- * of the ray (the plane) to the ray end.

regnpts/bottom/4/**pset,get,rayend/xyz**/0.,0.,-.1/0.,1.,-.1/1.,1.,-.1/1.,.6/



front face of cube showing one row of 'rayend' points and one set of 5 rays. Points are distributed in the region 'bottom'.

Other versions of the **regnpts** are appropriate for cylindrical and spherical geometries. For cylindrical geometries the **rz** command specifies points in a cylindrical shell outside the volume. The **regnpts** command specifies a line (usually the cylinder axis), and the rays are constructed normal to this line and containing one of the **rz** points. For spherical geometries the **rz** command specifies points in a spherical shell outside the volume. The **regnpts** command specifies a point (usually the center of the sphere) from which rays are constructed to the **rz** points.

If there are other regions that intrude on the region in which points are being distributed, then the effect is that of laying down a background distribution of points and erasing those that occur in the interior of the intruding regions.

7. Connect the points into tetrahedra

The mesh designer may use the following set of command to connect the points into a tetrahedral mesh:

- * eliminate coincident or nearly coincident points

- * 1,0,0 means consider all points

filter/1,0,0/

- * *rayend* points are set to invisible (**dud** is the code for invisible)

- * they were used as end points of the rays in the **regnpts** command

zq/itp/pset,get,rayend/dud/

- * assign material colors to the points

- * identify points that are on material interfaces

- * identify constrained points

setpts

- * connect the points into a Delaunay tetrahedral mesh

- * do not connect across material interfaces - add points if necessary to resolve material interfaces

search

- * set element (tetrahedral) type

- * spawn child points at material interfaces

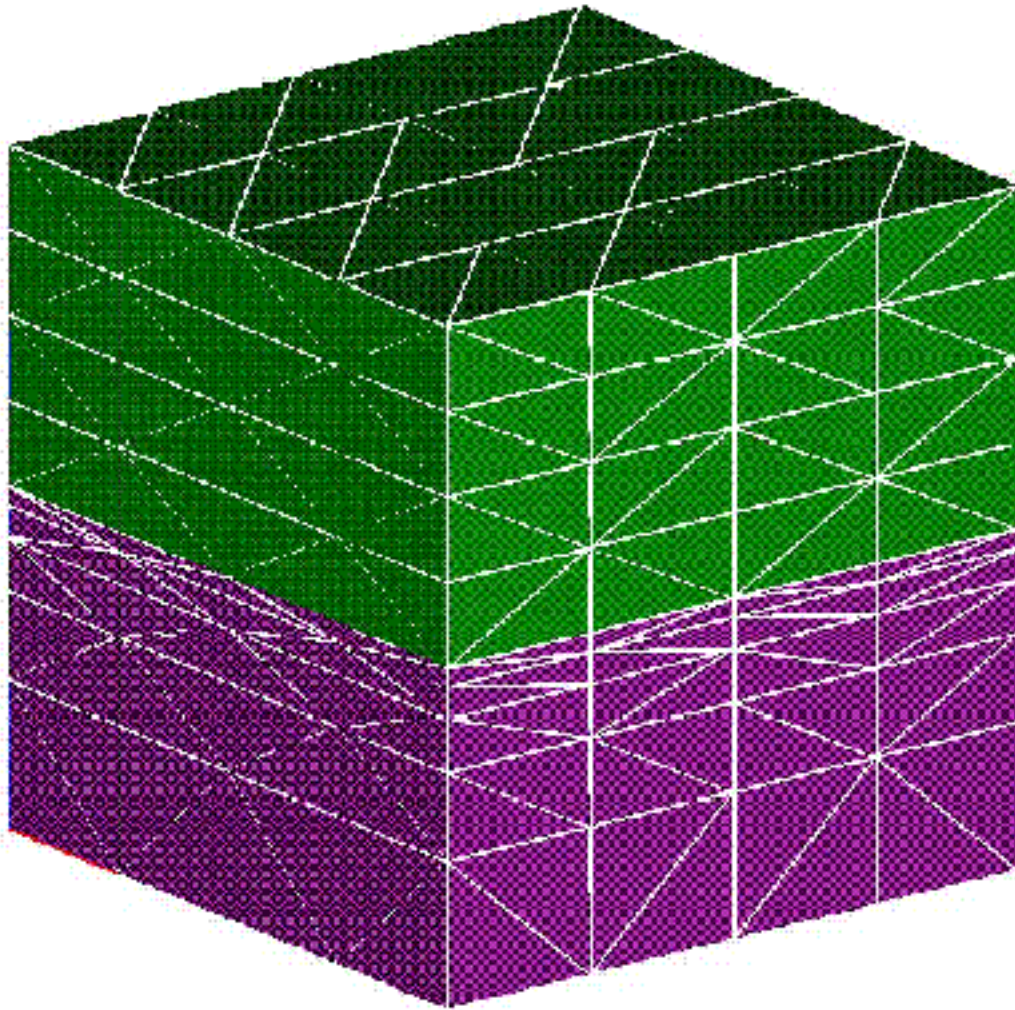
settets

- * dump mesh to some output form

dump/gmw/filename

- * terminate processing

finish



The complete input for the tutorial is:

```
* create a 3D tetrahedral mesh object and name it 3dmesh
cmo/create/3dmesh/
* unit cube
surface/cube/reflect/box/0.0,0.0,0.0/1.0,1.0,1.0/
* define z=.5 plane as interface
surface/cutplane/intrface/plane/0.,0.,.5/1.,0.,.5/1.,1.,.5/
*define geometric regions
region/top/ le cube and gt cutplane /
region/bottom/ le cube and le cutplane /
* define material regions
mregion/mattop/ le cube and gt cutplane /
mregion/matbot/ le cube and lt cutplane /
* create 25 points (5x5x1) in a plane above the unit cube
* place points on the boundaries in the x and y directions (1,1,0)
rz/xyz/5,5,1/0.,0.,1.1/1.,1.,1.1/1,1,0/
* give the points defined by the rz command the name, rayend
```

pset/rayend/**seq**/1,0,0/

- * create rays between points in *rayend* and the plane below the cube
- * distribute 3 points along these rays in the region *top*
- * add one point at the upper external boundary for each ray

regnpts/top/3/**pset,get**,rayend/**xyz**/0.,0.,-.1/0.,1.,-.1/1.,1.,-.1/0,0/

- * distribute 4 points along these rays in the region *bottom*
- * add one point at the lower external boundary for each ray
- * add one point at the material interface for each ray since
- * *bottom* contains the interface - a total of 5 points for each ray.
- * points will be distributed such that the ratio of distances between
- * any two consecutive pairs of points is 0.6 traveling from the source
- * of the ray (the plane) to the ray end.

regnpts/bottom/4/**pset,get**,rayend/**xyz**/0.,0.,-.1/0.,1.,-.1/1.,1.,-.1/1.,.6/

- * eliminate coincident or nearly coincident points
- * 1,0,0 means consider all points

filter/1,0,0/

- * *rayend* points are set to invisible (**dud** is the code for invisible)
- * they were used as end points of the rays in the **regnpts** command

zq/itp/pset,get,rayend/**dud**/

- * assign material colors to the points
- * identify points that are on material interfaces
- * identify constrained points

setpts

- * connect the points into a Delaunay tetrahedral mesh
- * do not connect across material interfaces -
- * add points if necessary to resolve material interfaces

search

- * set element (tetrahedral) type

settets

- * dump mesh to some output form

dump/gmw/filename

- * terminate processing

finish

II. Mesh Objects

a. *Mesh Object Definition*

The data structure which contains the information necessary to define a mesh is called a Mesh Object. A Mesh Object consists of attributes and parameters. There is a default template for a Mesh Object which consists of the following attributes:

name (mesh object name)

nnodes (number of nodes in the mesh)

nelements (number of elements in the mesh, e.g. triangles, tetrahedra)

nfaces (number of unique topological facets in the mesh, e.g. number of edges in 2D or number of element faces in 3D) -- (not used)

nedges (number of unique edges in mesh) -- (not used)

mbndry (value signifying that if the node number is greater than mbndry then the node is a boundary node)

ndimensions_topo (topological dimensionality, 1, 2 or 3, i.e. a non-planar surface would have ndimensions_topo = 2 and ndimensions_geom = 3.)

ndimensions_geom (1, 2 or 3 for dimension of geometry)

nodes_per_element

edges_per_element

faces_per_element (topological number of facets per element (i.e. in 1D this number is always 2, for 2D use the number of edges of the element, for 3D use the number of faces of the element.)

isetwd (integer array containing pset membership information, see **pset** command definition)

ialias (integer array of alternate node numbers, i.e. for merged points)

imt1 (integer array of node material)

itp1 (integer array of node type - type 20 node will be invisible)

<u>point type</u>	<u>name</u>	<u>description</u>
0	int	Interior
2	ini	Interface
3	vrt	Virtual
8	vif	Virtual + interface + free
9	alb	Virtual + Interface + free + reflective
10	rfl	Reflected boundary node
11	fre	Free boundary node
12	irb	Interface node on reflected boundary

13	ifb	Interface node on free boundary
14	rfb	Node on intersection of free boundary and reflective boundary
15	irf	Interface node on intersection of free boundary and reflective boundary
16	vrb	Virtual node on reflective boundary
17	vfb	Virtual node on free boundary
18	vfb	Virtual node on free + reflective boundary
19	vit	Virtual + interface node on reflective boundary
20	mrg	Merged node
21	dud	Dudded node
41	par	Parent node

icr1 (integer array of constraint numbers for nodes)

isn1 (integer array of child, parent node correspondence)

Points on material interfaces are given point type 41 (parent). One child point is spawned for each material meeting at the parent point. The isn1 field of the parent point will contain the point number of the first child point. The isn1 field of the first child will contain the point number of the next child. The isn1 field of the last child will contain the point number of the parent. The point types of the child points will be 2, 12 or 13 depending on whether the interface point is also on an exterior boundary. This parent, child relationship is established by the **settets** command.

ign1 (integer array of generation numbers for nodes)

xic, yic, zic (real arrays of node coordinates)

itetclr (integer array of element material)

itettyp (geometry of element)

<u>name</u>	<u>value</u>	<u>description</u>
ifelmpnt	1	point
ifelmlin	2	line
ifelmtri	3	triangle
ifelmqud	4	quadrilateral
ifelmtet	5	tetrahedron
ifelmpyr	6	pyramid
ifelmpri	7	prism
ifelmhhex	8	hexahedron
ifelmhgb	9	hybrid
ifelmply	10	polygon

xtetwd (real array containing eltset membership information, see **eltset** command definition)

itetoff (index into itet array for an element)

jtetoff (index into jtet array for an element)

itet (integer array of node vertices for each element)

jtet (integer array of element connectivity)

The default Mesh Object can be expanded by adding user defined attributes (see **cmo/addatt**). There are four special user defined attributes: velocity, density, pressure and energy; these attributes have pre-defined names which are stored as Mesh Object parameters. Most Mesh Object parameters are relevant only to the physics model, and the physics parameters will be documented in a later volume (a full list of all parameters will appear in the response to a **help** command under the heading: The CMO commands). The parameters relevant to grid generation are:

<u>Name</u>	<u>Default</u>	<u>Description</u>
densname	dens	Name of the user added density attribute
presname	pres	Name of the user added pressure attribute
enename	ener	Name of the user added energy attribute
velname	vels	Name of the user added velocity attribute
epsilon	10.e-15	
epsilon1	10.e-8	Minimum edge length
epsilona	10e-8	Minimum facet area
epsilonv	10e-8	Minimum element volume
ipointi	0	First point added by last generate step
ipointj	0	Last point added by last generate step
ipoints	0	Point stride of last generate step
var0,...var9	0	Available for user
itypconv_sm		Smoothing parameters see smooth
maxiter_sm		
tolconv_sm		

The value of parameters can be changed by the **assign** command.

X3D will add attributes to the mesh object in certain instances. For example, if there are any constrained surfaces, reflect, virtual or intrcons types, the following attributes are added to the mesh object:

NCONBND number of combinations of constrained surfaces

ICONBND(50,NCONBND)

ICONBND(1,i)	number of surfaces contributing to the ith constraint
ICONBND(2,i)	degree of freedom of the ith constraint
ICONBND(2+j,i)	Surface number of the jth surface contributing to the ith constraint

In order to determine which constraint entry applies to node ip, retrieve the value $i=icr1(ip)$, i.e. $ICONBND(1, icr1(ip))$ gives the number of surfaces that ip is 'on'. If $icr1(ip)$ is zero there is no constraint on that node.

TENSOR	Dimension of XCONTAB
XCONTAB(TENSOR,NPOINTS)	This is a 3x3 matrix which multiplied by the velocity vector, constrains the velocity to the number of degrees of freedom possessed by the node.

b. Command Interface

The default Mesh Object is named *3dmesh*. For simple problems the user must supply only a **cmo/create/mesh_object_name** command. There is no limit on the number of Mesh Objects that can be defined, but at any time there is only one 'current' or 'active' Mesh Object. For more advanced problems, such as those requiring more than one Mesh Object or requiring extensions to the basic Mesh Object template, the Mesh Object(s) is(are) manipulated via the **cmo** commands which are described in the next section. For example, additional user defined attributes may be added to a Mesh Object by using the **cmo/addatt** command, or the 'active' Mesh Object can be changed using the **cmo/select** command.

c. FORTRAN Interface

Mesh Object attribute data are accessed through a set of subroutines. An example of accessing an existing Mesh Object and creating a new mesh object is given in

Section IV; that example should be used as a template when operating with Mesh Objects. The subroutine set includes:

<code>cmo_get_name</code>	retrieve active mesh object name
<code>cmo_set_name</code>	set active mesh object name
<code>cmo_get_info</code>	retrieve mesh object data
<code>cmo_set_info</code>	set mesh object data
<code>cmo_newlen</code>	adjust mesh object memory allocation

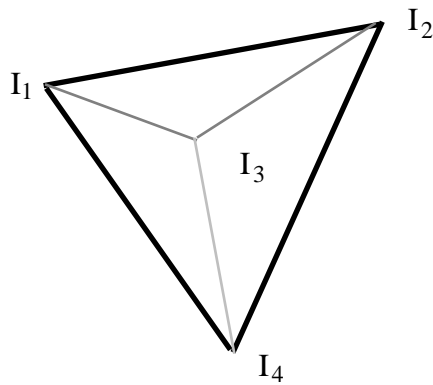
Only data from the active Mesh Object may be retrieved; calling `cmo_set_name` will make the Mesh Object active. Scalar quantities are retrieved and stored using `cmo_get_info` and `cmo_set_info`. Vector quantities are referred to by their pointers. The length of the vectors is calculated internal to X3D based on the scalar mesh object variables. Memory allocation for a new mesh object or for a mesh object which will grow in size is accommodated by setting the appropriate scalars for the Mesh Object and calling `cmo_newlen`. `cmo_set_info` with the new number of elements and/or nodes and `cmo_newlen` must be called before adding to the size of a Mesh Object.

Mesh object parameters are retrieved with the `get_info_c` subroutine.

See V.e.1 for a list of . mesh object subroutines .

d. Mesh Object Connectivity

The Mesh Object attributes, **itettyp**, **itetoff**, **jtetoff**, **itet**, and **jtet** along with the variables contained in the include file **local_element.h** completely describe the mesh connectivity. The following discussion is based on the concept of local facets and local edges for an element. The nodes comprising a given element are always specified in a well-defined order; hence when one references the 'second facet' of an element, one references a pre-defined set of points. Consider a tetrahedral element, with nodes labeled as in the diagram:



The points are oriented so that the triple product $I_1 I_2 \times I_1 I_3 \cdot I_1 I_4$ is positive, and the volume of the tet is one-sixth of the triple product. The local facets are defined as follows:

$$\begin{array}{lll} F_1 & = & I_2 \quad I_3 \quad I_4 \\ F_2 & = & I_1 \quad I_4 \quad I_3 \\ F_3 & = & I_1 \quad I_2 \quad I_4 \\ F_4 & = & I_1 \quad I_3 \quad I_2 \end{array}$$

The local edges for a tetrahedral are defined as follows:

$$\begin{array}{ll} E_1 & I_1 \quad I_2 \\ E_2 & I_1 \quad I_3 \\ E_3 & I_1 \quad I_4 \\ E_4 & I_2 \quad I_3 \\ E_5 & I_2 \quad I_4 \\ E_6 & I_3 \quad I_4 \end{array}$$

Similarly, local facets and local edges are defined for all element types.

itettyp(it) gives the element type of element **it**.

itetoff(it) gives the offset to the first node in element **it**.

itet(itetoff(it)+j) gives the jth node of element **it**.

nelmnen(itettyp(it)) gives the number of nodes of element **it**.

To loop through all the nodes of all elements in the mesh:

```
do it=1,ntets
  do j=1,nelmnen(itettyp(it))
    k=itet(itetoff(it)+j)
  enddo
enddo
```

nelmnef(itettyp(it)) gives the number of facets of element **it**.

ielmface0(iface,itettyp(it)) gives the number of nodes on facet **iface** of element **it**.

ielmface1(local_node,iface,itettyp(it)) gives the increment to the node number (**local_node**) on facet **iface** of element **it**.

To loop through all the nodes, **k**, of all elements in the mesh by facets:

```
do it=1,ntets
  do i=1,nelmnef(itettyp(it))
    do j=1,ielmface0(i,itettyp(it))
      k=itet(itetoff(it)+
        ielmface1(j,i,itettyp(it)))
    enddo
  enddo
enddo
```

nelmnee(itettyp(it)) gives the number of edges of element **it**.

ielmface2(inode,iface,itettyp(it)) gives the edge number associated with inode on facet **iface** of element **it**.

ielmedge1(1|2,iedge,itettyp(it)) gives the node offset associated with edge **iedge** of element **it**.

To loop through all pairs of edge nodes (**i1,i2**) of all elements in the mesh :

```
do it=1,ntets
  do i=1,nelmnee(itettyp(it))
    i1=itet1(itetoff(it)+
      ielmedge1(1,i,itettyp(it)))
    i2=itet1(itetoff(it)+
      ielmedge1(2,i,itettyp(it)))
  enddo
```

```
enddo
```

To loop through all pairs of edge nodes (**i1,i2**) of all elements in the mesh by facets:

```
do it=1,ntets
  do i=1,nelmnef(itettyp(it))
    do j=1,ielmface0(i,itettyp(it))
      ie=ielmface2(j,i,itettyp(it))
      i1=itet(itetoff(it)+
        ielmedge1(1,ie,itettyp(it)))
      i2=itet(itetoff(it)+
        ielmedge1(2,ie,itettyp(it)))
    enddo
  enddo
enddo
```

jtet(itetoff(it)+j) gives the element number and local facet number of the neighbor to element **it**, facet **j**.

To loop find all neighbors of elements (**jt** is neighbor element number, **jf** is facet of neighboring element), (**mbndry** is the value added to **jtet** if element **it** is on a boundary or interface; the **jtet** value of an element **it** with facet **j** on an exterior boundary will be exactly **mbndry**; the **jtet** value of an element **it** with facet **j** on an interior interface will be **mbndry** + the **jtet** value calculated from the neighboring element number and neighbor element local_facet number):

```
c set number of faces per element for this mesh object
call cmo_get_info('faces_per_element',cmo_name,
nef_cmo,ilen,ity,ics)
do it=1,ntets
  do i=1,nelmnef(itettyp(it))
c check if element face is on an external boundary
    if(jtet1(jtetoff(it)+i).eq.mbndry) then
      jt=0
      jf=0
c check if element face is on an internal boundary
    elseif(jtet1(jtetoff(it)+i).gt.mbndry) then
      jt=1+(jtet1(jtetoff(it)+i-mbndry-1))
    /nef_cmo
```

```

                                jf=jtet1(jtetoff(it)+i)-mbndry-1)+(
                                    nef_cmo*(jt-1)
C   Volume element
                                else
                                    jt=1+(jtet1(jtetoff(it)+i)-1/nef-cmo
                                    jf=jtet1(jtetoff(it)+i)-nef-cmo*(jt-1)

                                endif
                            enddo
                        enddo

```

III. X3D Commands:

a. *Conventions*

Following in Section III.b is list of the X3D commands. These commands are given in alphabetic order. Conventions that apply to all commands include:

1. Lines are a maximum of 80 characters long, identifiers are a maximum of 32 characters long.

2. Continuation lines are signaled by an "&" as the last character of a line to be continued. A command can be up to 1024 characters long.

3. Delimiters are comma, slash, equal sign, or blank. (',' '/' '=' ' ').

Blanks on either side of other delimiters are ignored. Leading blanks are ignored.

Commas are usually used for parameters that belong to the same logical set such as *first point*, *last point*, *stride*. Slashes are usually used to separate sets of parameters.

4. The three parameters: *first point*, *last point*, *stride* can have integer values which refer to actual sequential point numbers or they can have the character-string values:

pset, **get**, *name* where *name* has been defined by a previous **pset** command. The triplet: 1, 0, 0 refers to all points. The triplet: 0, 0, 0 refers to the set of points defined in the last geometry command.

5. Commands should be typed in lower case, however names are case sensitive.

In the command description that follows certain symbols have special meaning.

[] surround optional parameters

| signifies alternate choices

, or / separates parameters

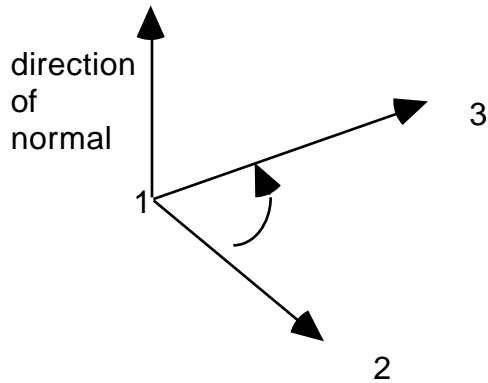
6. Courier font is used for variable names such as *ifirst*.

bold is used for literals such as **xyz**.

7. Comments are identified by * in the first column. Comments are parsed; avoid using special characters especially '&' in comments.

8. All names (*surface*, *region*, *pset*,...) should be limited to 32 characters.

9. The right hand rule is used to determine normals to planes and to sheet surfaces. The first two points determine the first vector and the first and third point determine the second vector. By curling the fingers of the right hand from the first vector toward the second vector, the right thumb will point in the direction of the normal.

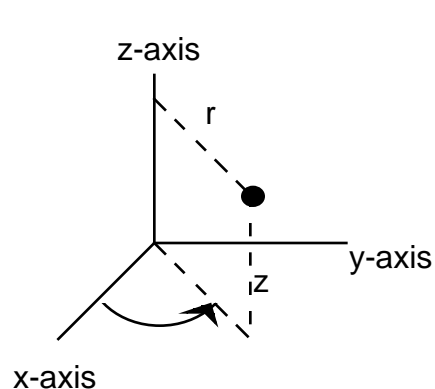


10. To separate commands on the same line use a semicolon (;).
11. Three coordinate systems are used.

xyz refers to the standard Cartesian coordinate system

rtz refers to a cylindrical coordinate system aligned along the z-axis, where **r** is the radius measured from the z-axis, **t** (theta) is the angle measured in the xy-plane from the positive x-axis toward the positive y-axis and **z** is the height measured from the xy-plane.

rtp refers to a spherical coordinate system, where **r** is the radius measured from the origin, **p** (phi) is the angle in the xy-plane measured from the positive x-axis toward the positive y-axis, **t** (theta) is the angle measured from the positive z-axis to the positive y axis.



cylindrical coordinates

rtz

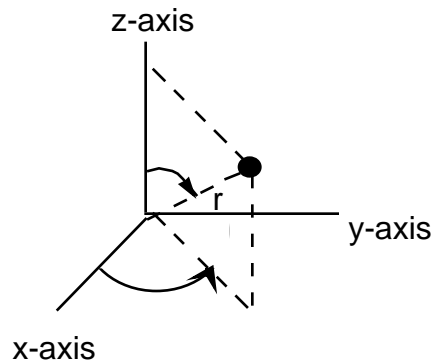
position of ●

determined by

r radius

t theta (angle from x-axis)

z height



spherical coordinates

rtp

position of ●

determined by

r radius measured from origin

t theta (measured from positive z-axis)

p phi (measured from positive x-axis)

b. Alphabetic Listing of X3D Commands

ADDMESH

This routine glues two meshes together at their common interface to produce a third mesh.

FORMAT:

addmesh / glue / mesh3 / mesh1 / mesh2 /

ASSIGN

Assign a value to a code variable.

FORMAT:

assign/category_name/column/variable_name/value.

EXAMPLES:

assign/-def/-def/epsilon/1.e-4/

CMO

The **cmo** command operates on the Mesh Object (MO). There can be many Mesh Objects in the code for a given problem. Only one of these Mesh Objects may be the Current Mesh Object. There is also one Default Mesh Object which is used as the template for generating new Mesh Objects.

FORMAT:

**cmo/addatt /mo_name/att_name/type/rank/length/interpolation/
persistence/io/value/
compress/mo_name/
copy/mo_name/master_mo/
create/mo_name/npoints/nelements/mesh_type/
create/mo_name/npoints/nelements/ndimensions_geom/ndimensions_topo/
nodes_per_element/faces_per_element/edges_per_element/
delatt/mo_name/att_name/
derive/mo_name/master_mo/
length/mo_name/att_name/
list
memory/mo_name/number_nodes/number_elements/**

modatt/mo_name/att_name/field_name/new_field/
move/mo_name/master_mo/
newlen/mo_name/
release/mo_name/
select/mo_name/
status/mo_name/
verify/mo_name/

CONVENTION: As a result of any command that generates a new Mesh Object, the newly generated Mesh Object becomes the Current Mesh Object.

RESERVED NAMES: The following names are reserved and may not be used for Mesh Object names:

-cmo-	the Current Mesh Object
-default-	the Default Mesh Object
-all-	all Mesh Objects or Attributes
-notset-	Null name for Mesh Object

TYPES ,DEFAULTS and POSSIBLE VALUES:

mo_name	is type character
att_name	is type character
mesh_type	is type character (tet,hex,pri,pyr,tri,qua,hyb)
type	is type character, default is VDOUBLE (VDOUBLE, INT, VINT)
rank	is type character, default is scalar (scalar,vector,tensor)
length	is type character, default is nnodes (nelements, nnodes)
interpolate	is type character, default is linear (copy, sequence, linear, log, asinh, max, min, user,and,or,incmax)
persistence	is type character, default is temporary (permanent, temporary)
ioflag	is type character, default is agx (a, g, f, s, x, no -- for avs,gmv,fehms,sgi,x3d)

default	is type real, default is 0.0
npoints	is type integer, default is Default MO
nelements	is type integer, default is Default MO
ndimensions_geom	is type integer, default is Default MO
ndimensions_topo	is type integer, default is Default MO
nodes_per_element	is type integer, default is Default MO
faces_per_element	is type integer, default is Default MO
edges_per_element	is type integer, default is Default MO

addatt/mo_name/att_name/type/rank/length/interpolate/persistence/io/value/

mo_name required

att_name required

Adds the Attribute, att_name, to Mesh Object, mo_name. See the **modatt** command for more details on the field variables.

Examples:

cmo/addatt/cm01/boron1/VDOUBLE/scalar/nnodes/asinh/permanent/gx/1.0

cmo/addatt/-cmo-/boron2/VDOUBLE/scalar/nnodes/asinh/permanent/gx/2.0

cmo/addatt/cm01/boron3/VDOUBLE/scalar/nnodes/user/temporary/gx/3.0

cmo/addatt/-default-/boron/VDOUBLE/scalar/nnodes/asinh/temporary/no/3.

cmo/addatt/-default-/boron3

compress/mo_name/

mo_name is type character, default is **-cmo-**

Shortens all memory managed arrays for Mesh Object mo_name to their actual lengths.

Examples:

cmo/compress/mo_tet2

cmo/compress/-cmo-

cmo/compress

cmo/compress/-all-

copy/mo_name/master_mo/

mo_name is type character, required.

master_mo is type character, default is **'-cmo-'**

Makes an exact copy of Mesh Object, master_mo, including all data. The output Mesh Object, mo_name, will become the Current Mesh Object. If mo_name is the same as master_mo nothing happens.

If mo_name exists it is over written.

Examples:

cmo/copy/mo_tet2/mo_tet1

cmo/copy/-cmo-/mo_tet1

cmo/copy/mo_tet2

cmo/copy/mo_tet2/-cmo-

create/mo_name/npoints/nelements/ **tet|hex|pri|pyr|tri|qua|hyb/**

or

create/mo_name/npoints/nelements/ndimensions_geom/ndimensions_topo/

nodes_per_element/faces_per_element/edges_per_element/

Creates a new Mesh Object 'mo_name', which becomes the Current Mesh Object.

If a Mesh is created using the first (mesh_type) format, then values are supplied for the

other parameters as follows:

mesh name	ndimension geom	ndimension topo	nodes_per element	faces_per element	edges_per element
tet	3	3	4	4	6
hex	3	3	8	6	12
pri (sm)	3	3	6	5	9
pyr (amid)	3	3	5	5	8
tri (angle)	3	2	3	3	3
qua (d)	3	2	4	4	4
hyb (rid)	3	3	10	10	12

If mo_name exists nothing happens.

mo_name required.

Examples

cmo/create/mo_tet2

cmo/create/mo_tet2/0/0/**hex**

delatt/mo_name/att_name/

Deletes the attribute att_name from Mesh Object, mo_name. Will not delete an attribute with a persistence of **permanent**

mo_name must be specified.

att_name must be specified.

Examples

cmo/delatt/mo_tet2/boron

cmo/delatt/-cmo-/boron

cmo/delatt/-default-/boron

cmo/delatt/-cmo/-all- (this will delete all attributes with persistence of **temporary**)

derive/mo_name/master_mo/

mo_name is type character, required.

master_mo is type character, default is **-cmo-**

Uses Mesh Object, master_mo, as the template for deriving Mesh Object,

mo_name. Mesh Object, mo_name, will be an image of master_mo but will

contain no data. The output Mesh Object, mo_name, will become the Current

Mesh Object. If mo_name is the same as master_mo nothing happens. If mo_name exists it is over written.

Examples:

cmo/derive/mo_tet2/mo_tet1

cmo/derive/-cmo-/mo_tet1

cmo/derive/mo_tet2

cmo/derive/mo_tet2/-cmo-

cmo/derive/-default/-cmo-

cmo/derive/mo_tet2/-default-

cmo/derive/-default-/mo_tet1

length/mo_name/att_name/

mo_name is type character, default is **-all-**

att_name is type character, default is **-all-**

Returns the memory length of attribute att_name for Mesh Object, mo_name.

Examples:

cmo/length/mo_tet2/boron

cmo/length/-cmo-/boron

cmo/length/-default-/boron

cmo/length/-cmo/-all-

cmo/length/mo_tet2/-all-

cmo/length

cmo/length/-all/-all-
cmo/length/-all-/boron

list

Returns the name of the Current Mesh Object and a list of all defined Mesh Objects

Examples:

cmo/list

memory/mo_name/number_nodes/number_elements/

Allows the user to preset the size of the memory managed arrays for Mesh Object ,
mo_name.

mo_name required.

number_nodes required.

number_elements required.

Examples:

cmo/memory/mo_tet2/1000/10000

cmo/memory/-cmo-/1000/10000

modatt/mo_name/att_name/field_name/new_field/

Modifies the field field_name for attribute att_name in Mesh Object mo_name.
mo_name required.

att_name required.

field_name is type character, required

new_field is the type of the field, required

Field_names (may be lower or upper case):

name - (character) Attribute name

type - (character) Attribute type:

INT- Integer

VINT - Vector of integer

VDOUBLE - Vector of real*8

rank - (character) Attribute rank (must be an attribute for this Mesh object)

length - (character) Attribute length (must be an attribute for this Mesh object)

interpolation - (character) Interpolation option:

constant	- Constant value
sequence	- Next is previous plus 1
copy	- Copy values
linear	- Linear interpolation
user	- User provides
log	- Logarithmic interpolation
asinh	- Asinh interpolation
persistence	- (character) Attribute persistence:
permanent	- Can not be deleted
temporary	- Temporary attribute
ioflag	- (character) Attribute IO flag:
a	- Put this attribute on avs dumps
g	- Put this attribute on gmV dumps
f	- Put this attribute on fehms dumps
s	- Put this attribute on sgi dumps
x	- Put this attribute on x3d dumps
no	- Ignore this attribute
default	(real) Attribute value

Examples:

```

cmo/modatt/mo_tet2/boron/length/nnodes
cmo/modatt/-cmo-/boron/length/nnodes
cmo/modatt/-cmo-/boron/default/10.0
cmo/modatt/-default-/boron/default/10.0

```

move/mo_name/master_mo/

mo_name is type character, required.

master_mo is type character, default is **-cmo-**

Changes the name of Mesh Object, master_mo, to mo_name. The output Mesh Object, mo_name, will become the Current Mesh Object. If mo_name is the same as master_mo nothing happens. If mo_name exists it is over written.

Examples:

```

cmo/move/mo_tet2/mo_tet1
cmo/move/-cmo-/mo_tet1
cmo/move/mo_tet2

```

cmo/move/mo_tet2/-cmo-

newlen/mo_name/

mo_name is type character, default is **-cmo-**

Changes the length of all memory managed arrays for Mesh Object mo_name to the proper length.

Examples:

cmo/newlen/mo_tet2

cmo/newlen/-cmo-

cmo/newlen

release/mo_name/

mo_name is type character, required.

Deletes the Mesh Object mo_name.

Examples:

cmo/release/mo_tet2

cmo/release/-cmo-

select/mo_name/

mo_name is type character, required.

Selects Mesh Object mo_name to be the Current Mesh Object. If mo_name does not exist a new Mesh Object will be created using the Default Mesh Object as the template.

Examples:

cmo/select/mo_tet2

status/mo_name/

mo_name is type character, default is **'-all-'**

Prints the status of Mesh Objects.

Examples:

cmo/status/mo_tet2

cmo/status/-cmo-

cmo/status
cmo/status/-all-
cmo/status/-default-

verify/mo_name/

mo_name is type character, default is '**-all-**'

Verifies that Mesh Object mo_name is consistent.

Examples:

cmo/verify/mo_tet2
cmo/verify/-cmo-
cmo/verify
cmo/verify/-all-
cmo/verify/-default-

COPYPTS

Copy a point distribution. There are two distinct forms of this command. The first format is designed to copy points from one mesh object into another. In this form if the names of the source and sink mesh objects are omitted, the current mesh object will be used. The copy may be restricted to a subset of points by including the source point information. Points in the sink mesh object will be overwritten if `sink_stride` is not zero. Attribute fields may be specified for both the source and sink mesh object. For example the x-coordinate field in the source mesh object (`xic`) may be placed in the y-coordinate field of the sink mesh object. Attribute values will be copied from the source mesh object to the sink mesh object. The user is warned that these values might not make sense in their new context.

The second form of this command is included for historic reasons: it duplicates points within a mesh object including all the attributes of the points. Also note that if no sink points or sink stride are specified, then the copied points are placed at the end of the data arrays (see third FORMAT) otherwise the copied points are written over the existing points starting at the 1st sink point. Note also that the first form of the command gives the arguments sink first then source whereas the second for give the source then the sink.

FORMAT:

copypts/sink_cmo/source_cmo/1st_sink_point/sink_stride /
1st_source_point/last_source_point/source_stride
/sink_attribute/source_attribute

copypts/1st_source_point/last_source_point/source_stride/
1st_sink_point/sink_stride /

copypts /1st_point/last_point/stride/

EXAMPLES:

copypts/3dmesh/2dmesh /

Copy all points in 2dmesh to the end of the 3dmesh point list.

copypts/3dmesh/2dmesh/0,0/**pset,get**,mypoints/

Copy the point set named *mypoints* from 2dmesh to the end of 3dmesh point list.

copypts/3dmesh/2dmesh/100,4/**pset,get**,mypoints/boron/arsenic/

Copy the arsenic field from the point set named *mypoints* from 2dmesh replacing the boron field at every fourth point beginning at point 100 in 3dmesh.

copypts/pset,get,mypoints/0,0/

Duplicate the point set named *mypoints* from the current mesh object and place the duplicated points at the end of the point list.

copypts///0,0/**pset,get**,mypoints/

Duplicate the point set named *mypoints* from the current mesh object and place the duplicated points at the end of the point list. Same effect as the example directly above. The current mesh object is used since the fields are blank on the command line

COORDSYS

This routine defines a local coordinate system to be in effect until another coordinate system is defined or the normal coordinate system is reset. The new coordinate system is defined by specifying an origin, a point on the new x-z plane and a point on the new z-axis. these points are specified in the normal coordinate system. the options available in *iopt* are:

define	define a new local coordinate system
normal	return to the normal coordinate system
save	save the current coordinate system for recall
restore	recall the last saved coordinate system

FORMAT:

coordsys/iopt/x0,y0,z0/xx,xy,xz/zx,zy,zz/

where x0,y0,z0 is the location of the new origin, xx,xy,xz is a point on the new x-z plane and zx,zy,zz is a point on the new z-axis. These points are defined with the normal coordinate system, and used only with the **define** option.

DOPING

Create doping profile for the grid.

A **constant** profile is invariant over the specified region with value **xcon**.

A **gaussian** profile contains a bounding box (x1,y1,z1) to (x2,y2,z2) where the peak concentration will be. z1=z2 in 2D. The doping varies according to the gaussian distribution:

$$\text{doping} = \text{concentration} * \exp(-(L/\text{std_dev})^2)$$

where L is the effective distance and can be represented as:

$$L = \sqrt{dy^2 + (1/\text{lateral_diffusion})*(dx^2 + dz^2)}$$

where

$$dy = y - y1 \text{ (or } y2 \text{ for that matter)}$$

$$dx = x - x1 \text{ if } x < x1 < x2$$

$$= 0 \text{ if } x1 < x < x2$$

$$= x - x2 \text{ if } x1 < x2 < x$$

$$dz \text{ similar to } dx.$$

The **table** option reads in doping data that has been read into the mesh object, **cmo_table_name**, as the attribute, **att_table_name** from a DATEX2.1 format file by a previously issued **read/datex** command. The fields **cmo_geometry** and **table_geometry** give the mapping from the domain on which the doping field was input to the active mesh object domain. The values of these fields may be **xy, yz, yx, xz, zx, zy** for 2-D geometries and **xyz, xzy, yxz, yzx, zxy, zyx** for 3-D geometries.

In all cases, **field** specifies the name of a defined attribute field in the active mesh object.

FORMAT:

doping/constant/field/set|add|sub/ifirst,ilast,istride/xcon

doping/gaussian/field/set|add|sub/ifirst,ilast,istride/xyz | rtz | rtp/

x1,y1,z1/x2,y2,z2/lateral_diffusion/concentration/std_dev/

doping/table/field/set|add|sub/ifirst,ilast,istride/cmo_table_name/

att_table_name/linear|log|asinh/cmo_geometry/table_geometry/

EXAMPLE:

```

doping/constant/ric/set/pset,get,Silicon/-1.0e+15/
doping/gaussian/ric/add/pset,get,Silicon/xyz/
    0.0,0.08,0.0/0.6,0.08,0.0/0.5/5.0e+18/0.225/
doping/table/pic/set/1,0,0/cmol/pic/linear/xyz/xyz/
doping/table/Saturation /set/1,0,0/cmo_course/Saturation/
    linear/zx/yx/

```

DUMP

This command produces an output file from a `Mesh` Object. If the option is **x3d**, a restart dump is made a subsequent **read/x3d** will restart the code at the state that the dump was taken.

FORMAT:

```

dump/file_type/file_name/[cmo_name]/
valid file_types are: x3d, gmw, avs, chad, feh, and datex

```

EXAMPLE:

```

dump/gmw/gmw.out/3dmesh/
dump/x3d/x3d.out/

```

EDIT

Prints an edit of various quantities based on the value of the option argument, the point limits, and/or a material specification. `iopt` specifies what to print as follows:

no value for `iopt` --edit of sums, averages, and extrema of position coordinates (x,y,z), and of mesh object attribute fields

two--gives same information as the default, but only for the two points specified.

parts--gives a list of materials types, their names, count and sequence.

points--lists up to 4 cell-center array values for a set of points. Possible array values are: `xic,yic,zic`, or mesh object attribute name

FORMAT:

```

edit/ iopt/ ifirst,ilast,istride/ material_#_or_name/
edit/ angular/ifirst,ilast,istride/material_#_or_name/xcen,ycen,zcen/
edit/ radial /ifirst,ilast,istride /material_#_or_name/xcen,ycen,zcen/

```

edit/ points /ifirst, ilast, istride /material_#_or_name/array1,array2 ,
array3,array4/
EXAMPLE:

edit/ parts/

edit/

edit/points/pset,get,some+points/Silicon/xic,yic,zic/

ELTSET

This command creates eltsets or element sets with membership criteria:

1. itetclr = , > , < value or other element attribute (added by cmo_addatt)
2. inclusive pset membership - all elements any of whose nodes is in pset
3. exclusive pset membership - all elements all of whose nodes are in pset
4. union, intersection, not, delete other eltsets
5. region or mregion membership

FORMAT:

eltset/eset_name/element_attribute_name/=| <|> | /value/

eltset/eset_name/**union**|**inter**|**not**|**delete**/eset_list/

eltset/eset_name/**inclusive**|**exclusive**/**pset/get**/pset_name/

eltset/eset_name/**region**|**mregion**/region_name|mregion_name/

EXAMPLE:

eltset/element_set1/**itetclr**/=/4

eltset/element_set2/**inclusive/pset/get**/point_set_1/

eltset/element_set3/**region**/upper_region/

EXTRACT

This command produces a 2D mesh object from a 3D mesh object. A material interface, a plane or an iso-surface may be extracted. A plane may be defined by three points in the plane, by a vector normal to the plane, by three points on the axes of the space, or by the coefficients of the plane equation $ax+by+cz=d$. An isosurface is defined by the value of the surface and the mesh object field to test for this value. An interface is defined by the material(s) bounding the interface. `region1`, [`region2`] are the material numbers or the material region names whose interface is to be extracted. Use **-a11-** to extract from all interfaces. All variations of the command can be limited by the usual pset syntax. The output 2D mesh object is `cmoout`, the input 3D mesh objects is `cmoin`.

The output MO will be oriented such that the outward normal of the plane that defines the surface will point in the same direction as the normals for the triangles in the output MO. If

the command extracts on an isosurface, the output MO will be oriented such that the normals for the triangles point in the direction of increasing field. If the command extracts on an interface, the output MO triangles will be oriented the same as the triangles extracted from region1 of the input MO. In the case of a plane extracted along all or a portion of a material interface, only those points that lie inside the material (i.e.: away from the direction of the normal) will be picked up. If the extraction is on a boundary, the normal to the extraction plane must point out of the material in order for points to be picked up.

FORMAT:

```
extract/plane/threepoints/x1,y1,z1/x2,y2,z2/x3,y3,z3/
                                ifirst,ilast,istride/cmooout/cmoin
/ptnorm/x1,y1,z1/xn,yn,zn/ ifirst,ilast,istride/cmooout/cmoin
/axes/xv,yv,zv/ ifirst,ilast,istride/cmooout/cmoin
/abcd/a,b,c,d/ ifirst,ilast,istride/cmooout/cmoin
/isosurf/var/value/ ifirst,ilast,istride/cmooout/cmoin
/interface/region1/ ifirst,ilast,istride/cmooout/cmoin
/intrfac2/region1/region2/ ifirst,ilast,istride/cmooout/cmoin
```

FIELD

The FIELD Command option manipulates one or more specified fields in the Current Mesh Object

- For all points in the specified point set, we **compose** the field value with the specified composition function. The composition functions allowed are currently **asinh** and **log**. So, for example, if 'i' is in the point set and **asinh** is the composition function, we have the assignment:

```
field(i) = asinh(field(i)).
```

- The **field/mfedraw** command causes a binary dump of the specified fields to two files in the **mfedraw** input format. **mfedraw** is a graphics package for visualizing moving piecewise linear functions of two variables, such as those originally encountered in Moving Finite Elements. The files are named 'root1.bin' and 'root2.bin', where 'root' is the root file name argument. Because the graphics data are a function of two variables, you must supply two orthonormal vectors (x1,y1,z1) and (x2,y2,z2) which specify the graphics coordinate axes. More precisely, given 3D coordinates (x,y,z), the

2D graphic coordinates will then be $(x*x1+y*y1+z*z1, x*x2+y*y2+z*z2)$. So, for example, the choice:

`/x1,y1,z1/x2,y2,z2/ = /1.,0.,0./0.,1.,0./`

causes the 'z' coordinate to be discarded while the 'x' and 'y' coordinates are unchanged.

- The **field/scale** option scales the field values of the specified points. `scale` option can take on the values **normalize**, **multiply**, and **divide**. If **normalize** is specified, we multiply all the field values by `factor/(fieldmax-fieldmin)`, where 'fieldmax' and 'fieldmin' are the maximum and minimum values taken over the point set. This has the effect of normalizing the field so that the new difference between the maximum and minimum values is equal to `factor`. If **multiply** is specified, we multiply all the field values in the point set by `factor`. If **divide** is specified, we divide all the field values in the point set by `factor`.
- The **field/volavg** option, for all the members of the point set and for all specified fields, replaces the point field values with values that represent the average of the field(s) over the control volumes associated with the points. The `averaging` option specifies what kind of control volume is to be used; the choices are **voronoi** and **median**. `iterations` is an integer that specifies a repeat count for how many times this procedure is to be performed on the field(s). The affect of this process is to broaden and smooth the features of the field(s), similar to the effect of a diffusion process. The **voronoi** choice, unlike the **median** choice, produces a diffusive effect independent of mesh connectivity. However, again unlike the **median** choice, it requires that the mesh be Delaunay, or incorrect results will occur.

FORMAT:

field/compose/composition function/ifirst,ilast,istride/field list/

field/mfedraw/root file name/x1,y1,z1/x2,y2,z2/field list/

field/scale /scale option/factor/ifirst,ilast,istride/ field list /

field/volavg/averaging option/iterations/ifirst,ilast,istride/field list/

EXAMPLE:

field/compose/asinh/1,0,0/pressure/

field/scale/normalize/4.0/set,get,region1/boron/

field/volavg/voronoi/4/1,0,0/boron/

FILTER

Used to filter (delete) points that are too close (default==> `tolerance=5.0E-06`), closer than the `tolerance` specified by the user, or duplicate points. This command records the deleted points as duddled out points (`itp=21`) and places their position at infinity but does not remove them from the point list. Note that at least one point must be specified in the point sequence numbers (`ifirst, ilast, istride`) in order for this command to work properly.

FORMAT:

filter / `ifirst, ilast, istride` / [`tolerance`]

FINISH

Terminate processing this set of command and return to the driver routine.

FORMAT:

finish

GENIEE

Generate element connectivity list(`jtet`) that gives neighbor information. Element connectivity is maintained by X3D, but can also be generated by the user with this command.

FORMAT:

geniee

HELP

Access help package. **help**/`command` will return the command description.

help/`code_variable` will return the variable definition. **help** with no arguments will return a list of commands and variables.

FORMAT:

help / [`variable_name` | `command_name`]

EXAMPLES:

help

help/surface

help/ipointi

HEXTOTET

Create a tetrahedral grid from a hexahedral grid or a triangle grid from a quadrilateral grid. **ioption** determines how the conversion is performed.

ioption = 2 ==> 2 triangles per quad, no new points.

ioption = 4 ==> 4 triangles per quad, 1 new point per quad.

ioption = 5 ==> 5 tets per hex, no new points.

ioption = 6 ==> 6 tets per hex, no new points.

ioption = 24 ==> 24 tets per hex, 7 new points(1 + 6 faces).

FORMAT:

hextotet/ioption/cmo_tet/cmo_hex/

EXAMPLES:

hextotet/24/cmo_tet/cmo_hex/

INFILE

INPUT

These commands instruct X3D to begin processing commands from a file. The **infile** commands may be nested. Only the outermost set of commands should be terminated with a **finish** command

FORMAT:

infile/file_name

input/file_name

INTERSECT

Creates a new Mesh Object from the intersection of two existing Mesh Objects. The existing Mesh Objects have to be topologically 2D and geometrically 3D. The created Mesh Object will be topologically 1D and geometrically 3D. Node quantities for the new Mesh Object will be created by interpolation on the corresponding node quantities of the first input Mesh Object, **cmo_1_in**.

FORMAT:

intersect/cmo_out/cmo_1_in/cmo_2_in

LOG

Turn the batch output file and tty output file **off** and **on**. The tty prints to and reads from the user's screen. The batch file is the output file called outx3dgen. Default is **on** for both files.

FORMAT:

log/bat|tty/on|off/

EXAMPLE:

log/tty/off

MERGE

Merge two points together. On return, the `first_point` is the survivor unless `first_point` may not be removed (a corner point for example), then the command operates as if `first_point` and `second_point` have been interchanged. If there is no confirmation of the merge, one or both of the points may be inactive, or the merge may be illegal because the points are not neighbors or because this merge is disallowed by the merge tables. Merging may trigger other merges by the reconnection step that follows the merge.

FORMAT:

merge/first_point/second_point/

EXAMPLE:

merge/21,22/

MREGION

Define a material region from a set of surfaces by logically combining the surface names and region names. The operators **lt**, **le**, **gt**, and **ge** are applied to previously defined surfaces according to the following rules.

lt -- if the surface following is a volume then **lt** means inside not including the surface of the volume. If the surface is a plane or a sheet **lt** means the space on the side of the plane or sheet opposite to the normal not including the plane or sheet itself.

le -- if the surface following is a volume then **le** means inside including the surface of the volume. If the surface is a plane or a sheet **le** means the space on the side of the plane or sheet opposite to the normal including the plane or sheet itself.

gt -- if the surface following is a volume then **gt** means outside not including the surface of the volume. If the surface is a plane or a sheet **gt** means the space on the same side of the plane or sheet as the normal not including the plane or sheet itself.

ge -- if the surface following is a volume then **ge** means outside including the surface of the volume. If the surface is a plane or a sheet **ge** means the space on the same side of the plane or sheet as the normal including the plane or sheet itself.

The operators **or**, **and**, and **not** applied to regions or surfaces mean union, intersection and complement respectively. The operators **or**, **and**, and **not** applied to relational operators are the normal logical operators. Parentheses are used for nesting. Spaces are required as delimiters to separate operators and operands. Internal interfaces should be excluded when defining material regions. (i.e. use **lt** and **gt**). External boundaries should be included when defining material regions. If a material regions consists of more than one region and the regions touch (i.e. share a region interface), then the region interface is not a material interface -- all the points on the region interface are interior to the material region. In this case use **le** or **ge** to include these region interface points in the material region as interior points.

FORMAT:

mregion/material_region_name/region definition

EXAMPLES.:

mregion/mat1/ **le** box1 **and** (**lt** sphere1 **and** (**lt** plane1 **or** **gt** plane2)) /

mregion/mat2/ regiona **or** regionb /

OFFSETSURF

Offsets triangulated surfaces in the direction of the surface outward normal, i.e., normal surface motion. For each node a 'synthetic' unit outward normal **N** is computed based on the weighted average angle of the normals of the triangles shared by that node. `old_cmo` is the surface to be used in generating the offset surface. `new_cmo` is the name of the new surface.

To add the nodes in the new surface to the main mesh object use the **copypts** command. To add the new surface to the main mesh object use a **surface** command with `new_cmo` as the sheet name (e.g. **surface**/s2d/bndy_type/sheet/new_cmo/).

`dist` is given in user coordinates (i.e. whatever units the `old_cmo` mesh object was defined in.)

The new node coordinates, `R_new`, are computed using the formula:

$$R_new = R_old + dist * N_node$$

FORMAT:

offsetsurf/new_cmo/old_cmo/dist

PSTATUS

Saves, removes, retrieves, or replaces a specified set of points, usually the last set of points defined by a generator command or the set of points defined by `ifirst, ilast, istride`. Note that point sets must be specified in sequence in order for this command to work properly.

FORMAT:

pstatus (Returns current point status counters)

pstatus /save/name/ifirst, ilast, istride/

(Saves the point status numbers, `ifirst, ilast, istride` under `name`)

pstatus /store/name/ifirst, ilast, istride/

(Overwrites what was in `name` with `ifirst, ilast, istride`)

pstatus /delete/name (Deletes values from `name`)

pstatus /get/name (Retrieves values from `name`)

PSET

Give a name to a selected set of points.

union, **inter** and **not** are logical operations on previously defined psets.

list lists all psets

delete deletes a previously defined pset

zq forms a pset from all points in `ifirst, ilast, istride` which have value `flag` for the attribute `point flag`.

geom forms a pset from all points inside the box whose corners are `x1, y1, z1` and `xu, yu, zu` relative to the geometry center at `xc, yc, zc`.

FORMAT:

pset/pset name/

seq/ifirst, ilast, istride

union|inter|not|delete/pset1[/pset2/.../psetn]
list
zq/point flag/ifirst,ilast,istride/flag
region/region name/ifirst,ilast,istride
geom/xyz/ifirst,ilast,istride/xl,y1,z1/xu,yu,zu/xc,yc,zc

EXAMPLE:

pset/apset/seq/1,0,0/ (give all points the name apset)
pset/apset/seq/0,0,0/ (give the last defined points the name apset)
pset/apset/union/pset1,pset2,pset3/ (combine psets)
pset//list/ (list all psets)
pset/apset/zq/imd/1,0,0/air/ (give all points in material region air the name apset)

QUADXY

Define an arbitrary, logical quad of points in 2D(xy) space. n_x and n_y specify the number of points in the x and y directions. The four corners of the quad are then listed in counter clockwise order (the normal to the quad points is defined using the right hand rule and the order of the points).

FORMAT:

quadxy/ n_x, n_y /x1,y1,z1/x2,y2,z2/x3,y3,z3/x4,y4,z4/

QUADXYZ

Define an arbitrary, logical hexahedron of points in 3D(xyz) space. n_x , n_y and n_z specify the number of points in the x and y and z directions. The eight corners of the hex are then listed as two sets of quads, each set of four nodes is given in counter clockwise order . Points 1 to 4 specify one face of the hex, points 5 to 8 the corresponding face opposite (point 5 is logically behind point 1, point 6 behind point 2 and so on.)

FORMAT:

quadxyz/ n_x, n_y, n_z /x1,y1,z1/x2,y2,z2/x3,y3,z3/x4,y4,z4/x5,y5,z5/x6,y6,z6/x7,y7,z7
 /x8,y8,z8/

READ

This command reads in data into the active Mesh Object, replacing whatever data might have been previously contained in the active Mesh Object. If the option is **x3d** the code reads in a restart dump. The **avs** option includes the choice of reading in nodes, elements and attributes by giving flags values of 1 (read) or 0 (skip) for the categories: node_flag, element_flag and attribute_flag. This option requires that the mesh object name be specified.

FORMAT:

```
read/avs|dcm|datex|x3d/file_name  
read/avs/file_name/cmo_name/node_ flag/element_flag/attribute_flag/  
read/x3d/file_name/[dump_name/region_name/]  
read/ngp/tet|hex|quad|tri/file_name/
```

EXAMPLE:

```
read/x3d/myfile
```

RECON

This command flips connections in the mesh to get restore the Delaunay criterion. The default is to add points on the boundaries if needed (**yes**). The option **no** specifies that no points are to be added on the boundaries.

FORMAT:

```
recon/[yes|no/]
```

REFINE

The **refine** command is used to create more elements when a criterion is specified. These criteria are defined in the Grid Refinement and Derefinement Section which follows.

The choice to refine is based on one of the following refine_criterion:

junction will refine object where field = crosses x refine

constant will refine object where field > xrefine

delta will refine object where field > xrefine

lambda will refine object where $\lambda(\text{field}) < \text{xrefine}$

maxsize will refine object where $\text{object} > \text{xrefine}$

maxsize refers to volume for tets, area for face, length for edges

minsize will derefine object where $\text{object} < \text{xrefine}$, (not implemented)

aspect will refine where $\text{aspect ratio} < \text{xrefine}$

addpts will refine explicitly by adding in a set of nodes

The `refine_type` specifies what object will be refined and how that object will be refined:

tet will refine elements by placing a point in the center of the element.

face will refine facets by placing a point in the center of the facet.

edge will refine edges by placing a point on the midpoint of the edge.

faceedge will refine facets by refining all edges of the facet.

tetedge will refine elements by refining all edges of the element.

The `field` must refer to a previously defined attribute of the current Mesh Object.

The `interpolation` specifies how to interpolate the field to give field values to the new nodes created. The implemented values are.

linear

log

asinh

The `inclusion_flag` specifies if refinement is an inclusive or an exclusive operation. If for example, an edge refinement is specified restricted to a pset, then an edge is eligible for refinement if either or both of the end points belong to the pset if the `inclusion_flag` is set to **inclusive**. If the `inclusion_flag` is **exclusive** then both end points must be in the pset. The implemented values are.

inclusive

exclusive

FORMAT:

```
refine/refine_criterion/field/interpolation/refine_type  
/ifirst,ilast,istride /xrefine/inclusion_flag
```

EXAMPLES:

```
refine/maxsize///edge/pset,get,something/.25/
```

```
refine/constant/concentration/log/edge/1,0,0/25./inclusive
```

`refine/addpts///tet/pset,get,newpoints/`

Grid Refinement and Derefinement Criteria

The Refine command for the grid generation code X3D uses various criteria to tag grid elements for refinement or derefinement. When utilizing unstructured grids generated by X3D for applications such as solution of partial differential equations (PDE) for physical systems, it is desirable to modify the grid in order to optimize it for the particular problem based on several principles. The goal is to produce a better solution by creating better grid elements in various regions of the domain of the PDE's. This can involve physical criteria such as choosing smaller elements where physical variables are rapidly changing or larger elements where the variables are relatively constant in order not to waste computational effort. Grid elements can also be chosen on various geometric criteria related to their shape such as different formulations of an aspect ratio. In time dependent problems, it may be necessary to refine and derefine the grid after each time step in order to follow various changing phenomena such as moving concentration fronts, shock waves, or advancing oxide layers. These factors make mesh refinement crucial to the practical solution of physical modeling problems. We therefore will detail several algorithms that are currently implemented for identifying which grid elements should be refined or derefined based on geometric and physical criteria. New algorithms will be added to this list and the current ones modified as we obtain feed back from users.

Grid Refinement and Derefinement Criteria - Algorithms

I. Edges: Each edge is tested separately to see if it should be tagged for refinement or derefinement.

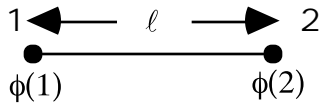
Definition:

χ = a user supplied tolerance

(i) = value of the field variable at node i

ℓ = length of the edge

For the edge between nodes 1 and 2, we have



Criteria:

1) Junction: Refine if the edge's field values straddle χ .

Tag for refinement if

$$\phi(1) > \chi \quad \text{and} \quad \phi(2) < \chi$$

or

$$\phi(1) < \chi \quad \text{and} \quad \phi(2) > \chi$$

example: For $\chi = 0$, refine if ϕ changes sign across the edge.

2) Constant: Refine if the edge's field values exceed χ .

Tag for refinement if

$$\phi(1) > \chi \quad \text{or} \quad \phi(2) > \chi$$

3) Maxsize: Refine if the edge length exceeds χ .

Tag if $\ell > \chi$

4) Minisize: Derefine if the edge length is smaller than χ .

Tag if $\ell < \chi$

5) Delta: Refine if the magnitude of the difference of the field values at the edge ends exceeds χ .

$$\text{Tag if } |\phi(1) - \phi(2)| > \chi$$

6) Lambda Refine: Refine if $\ell / X < \chi$. Where X is a scale length (here taken to be ℓ) and χ is given below.

$$\lambda = \frac{|\phi(\bar{x}_c)|}{|\bar{\phi}|}$$

where \bar{x}_c is the location of the edge center.

$$|\bar{\phi}| = \frac{|\phi(1) - \phi(2)|}{\ell}$$

$$|\phi(\bar{x}_c)| = \frac{1}{2} |\phi(1) + \phi(2)|$$

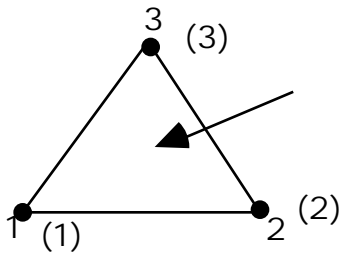
$$\text{Tag if } \frac{1}{2} \frac{|\phi(1) + \phi(2)|}{|\phi(1) - \phi(2)|} < \chi$$

Generally λ / X is a quality measure of the discretization. A larger value of λ / X usually indicates a better grid discretization. There are some special cases however. If one of the field values ϕ is zero as could happen on a boundary, then $\lambda / X = 1/2$ always. Another special case would be for $\phi(1) = \phi(2)$ then λ / X is divergent but the algorithm uses a small number $\epsilon = 1 \times 10^{-6}$ added to the denominator to prevent this to give a large but finite value of λ / X thus indicating a good discretization.

7) Lambda Derefine: As above for Lambda Refine except tag for derefinement if

$$\bar{\lambda} >$$

II. Faces: Each face is tested separately for refinement or derefinement. For the tetrahedral face defined by nodes 1, 2, and 3, we have



where A is the area of the face.

Criteria:

- 1) Junction: Refine if any of the faces' field values straddle .

Tag for refinement if

$$\phi(1) > \chi \quad \text{and} \quad \phi(2), \phi(3) < \chi$$

or

$$\phi(2) > \chi \quad \text{and} \quad \phi(1), \phi(3) < \chi$$

or

$$\phi(3) > \chi \quad \text{and} \quad \phi(1), \phi(2) < \chi$$

or all of the above with > and < interchanged.

example: For $\chi = 0$, refine if ϕ changes sign between any of the three nodes.

- 2) Constant: Refine if any of the faces' field values exceed .

Tag for refinement if

$$\phi(1) > \chi \quad \text{or} \quad \phi(2) > \chi \quad \text{or} \quad \phi(3) > \chi$$

- 3) Maxsize: Refine if the face area exceeds .

Tag if $A >$

- 4) Minisize: Derefine if the face area is less than .

Tag if $A <$

- 5) Aspect Ratio: Refine if the face's aspect ratio is less than . The aspect ratio (AR) is defined as the ratio of the radius of the inscribed circle of the triangular face to the radius of the circumscribed circle. We renormalize this ratio of multiplying by 2 so that the ratio equals one for an equilateral triangle.

$$AR = 2 \frac{R_c}{R_C} \quad \text{where } R_c \text{ radius of inscribed circle}$$

R_C radius of circumscribed circle

AR is never greater than one.

Tag if $AR <$

Generally the smaller AR is the more elongated the face is.

- 6) Lambda Refine: Refine if $\lambda / X < \chi$. Where X is taken to be the radius of the circumscribed circle R_C of the triangular face

$$\lambda = \frac{|\phi(\bar{X}_c)|}{|\bar{\phi}|}$$

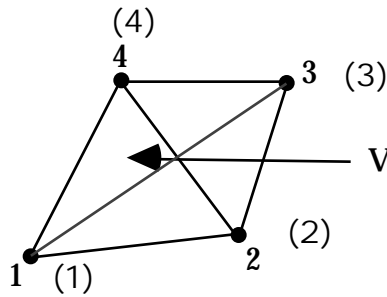
$$|\phi(\bar{X}_c)| = \frac{1}{3}|\phi(1) + \phi(2) + \phi(3)|$$

where \bar{X}_c is the centroid of the triangular face, and we have assumed a linear interpolation of ϕ .

$|\bar{\phi}|$ is evaluated on the face by a suitable approximation involving a linear interpolation of ϕ and $|\bar{\phi}| = \frac{1}{A} \oint \phi(\bar{X}) d\bar{\ell}$ where the line integral is around the edge of the face.

- 7) Lambda Derefine: As above for Lambda Refine except tag for derefinement if $\frac{\lambda}{X} > \chi$

III. Tets: Each tetrahedron is tested separately for refinement or derefinement



where V is the volume of the tetrahedron.

Criteria:

- 1) Junction: Refine if any of the tet's field values straddle χ .
example: For $\chi = 0$ refine if ϕ changes sign between any of the four nodes.
- 2) Constant: Refine if any of the tet's field values exceed χ .
Tag for refinement if
 $\phi(1) > \chi$ or $\phi(2) > \chi$ or $\phi(3) > \chi$ or $\phi(4) > \chi$
- 3) Maxsize: Refine if the tet volume exceeds χ .
Tag if $V > \chi$
- 4) Minisize: Derefine if the tet volume is less than χ .
Tag if $V < \chi$
- 5) Aspect Ratio: Refine if the tet's aspect ratio is less than . For the tet the aspect ratio (AR) is deferred as the ratio of the radius of the inscribed sphere of the tet to the radius of the circumscribed sphere. We renormalize this ratio by multiplying by three so that the ratio equals one for a regular tetrahedron (composed of equilateral triangular faces).

$$AR = 3 \frac{R_s}{R_S} \quad \text{where} \quad R_s \quad \text{radius of inscribed sphere}$$

R_S radius of circumscribed sphere

AR is never greater than one.

Tag if $AR < .$

Generally the smaller AR is, the more elongated the tet is.

- 6) Lambda Refine: Refine if $/ X < .$ Where X is taken to be the radius of the circumscribed sphere R_S of the tet.

$$\lambda = \frac{|\phi(\bar{X}_c)|}{|\bar{\phi}|}$$

$$|\phi(\bar{X}_c)| = \frac{1}{4} |\phi(1) + \phi(2) + \phi(3) + \phi(4)|$$

where \bar{X}_c is the centroid of the tet, and we have assumed linear interpolations of ϕ . $|\bar{\phi}|$ is evaluated for the tet by a suitable approximation involving a linear interpolation of ϕ and $|\bar{\phi}| = \frac{1}{4} \oint \phi(\bar{X}) d\bar{s}$ where the surface integral is over the surface of the tet.

7) Lambda Derefine: As above for Lambda Refine except tag for derefinement if $\frac{\lambda}{\lambda_{ref}} >$

IV. Face Edges: Same algorithms of the Faces category except all edges of the face are tagged for refinement or derefinement if the condition is met for the face.

V. Tet Edges: Same algorithms of the tet category except all edges of the tet are tagged for refinement or derefinement if the condition is met for the tet.

REGION

Define a geometric region from the set of surfaces by logically combining the surface names. The operators **lt**, **le**, **gt**, and **ge** are applied to previously defined surfaces according to the following rules.

lt -- if the surface following is a volume then **lt** means inside not including the surface of the volume. If the surface is a plane or a sheet **lt** means the space on the side of the plane or sheet opposite to the normal not including the plane or sheet itself.

le -- if the surface following is a volume then **le** means inside including the surface of the volume. If the surface is a plane or a sheet **le** means the space on the side of the plane or sheet opposite to the normal including the plane or sheet itself.

gt -- if the surface following is a volume then **gt** means outside not including the surface of the volume. If the surface is a plane or a sheet **gt** means the space on the same side of the plane or sheet as the normal not including the plane or sheet itself.

ge -- if the surface following is a volume then **ge** means outside including the surface of the volume. If the surface is a plane or a sheet **ge** means the space on the same side of the plane or sheet as the normal including the plane or sheet itself.

The operators **or**, **and**, and **not** applied to surfaces mean union, intersection and complement respectively. The operators **or**, **and**, and **not** applied to relational operators are the normal logical operators. The parentheses operators, (**and**), are used for nesting. Spaces are required as delimiters to separate all operators and operands. Internal interfaces should be included in exactly one region.

FORMAT:

region/region_name/region definition

EXAMPLES:

region/reg1/**le** sphere1 **and** (**lt** plane1 **or** **gt** plane2)

region/reg2/**le** sphere1 **and** (**ge** plane1 **and** **le** plane2)

REGNPTS

Generates points in a region previously defined by the **region** command. The points are generated by shooting rays through a user specified set of points from an origin point, line, or plane and finding the intersection of each ray with the surfaces that define the region. The point distribution is determined by the data in `ptdist`. If `ptdist` is integer, then that many points are evenly distributed along the ray in the region. If `ptdist` is real,

then points are distributed at that distance along the ray within the region. The variables `irratio` and `rrz` determine ratio zoning when `ptdist` is an integer. Ratio zoning is on when `irratio` is 1, then the distribution is adjusted by the value for `rrz`. When `irratio` is 2, the points are distributed by equal volumes depending on the geometry. When `irratio` is 3, ratio zoning is calculated on the longest ray then this length distribution is applied to all the rays. See the description of the command **surface** for a discussion of point distributions with respect to sheet surfaces.

FORMAT:

```
regnpnts/region name/ptdist/ifirst,ilast,istride/geom/  
ray origin/irratio,rrz  
regnpnts/region name/ptdist/pset,get,setname/geom/ray origin  
/irratio,rrz
```

Where `ifirst,ilast,istride` or **pset,get,setname** define the set of points to shoot rays through.

EXAMPLES:

```
regnpnts/region name/ptdist/ifirst,ilast,istride/xyz  
/x1,y1,z1/x2,y2,z2/x3,y3,z3/irratio,rrz/
```

Where points 1, 2, 3 define the plane to shoot rays from that are normal to the plane.

```
regnpnts/region name/ptdist/ifirst,ilast,istride/  
rtz/x1,y1,z1/x2,y2,z2/irratio,rrz/
```

Where points 1, 2, define the line from which to shoot perpendicular rays

```
regnpnts/region name/ptdist/ifirst,ilast,istride/  
rtp/xcen,ycen,zcen/irratio,rrz
```

Where `xcen,ycen,zcen` define a point from which to shoot rays .

```
regnpnts/region name/ptdist/ifirst,ilast,istride/points/  
iffirst,iflast,ifstride/
```

Where `iffirst,iflast,ifstride` define a set of points from which to shoot rays

RESETPTS

Reset node values. If `iflag` is parent (default) the parent child flags are reset. All child points are eliminated and the connectivity list is corrected to reference only the parent points. If `iflag` is `itp` the `itp1` array is reset to indicate whether each node is in the interior (0), on an interior interface (2), on a reflected boundary (10), or on a reflected interface boundry (12) . Resetting `itp` would be used if nodes were removed (such as with `rmnat`) leaving new boundaries

FORMAT:

resetpts/iflag

EXAMPLES:

resetpts

resetpts/parent

resetpts/itp

RM

Removes any points that are within the specified point range and specified volume of space. This is done in Cartesian (X, Y, Z), cylindrical (R, THETA, Z), or spherical (R, THETA, PHI) coordinates. It should be noted that in cylindrical coordinates, `theta` is the angle in the XY- plane with respect to the x-axis, while in spherical coordinates `theta` is the angle with respect to the Z-axis and `phi` is the angle in the XY-plane with respect to the X-axis. In cylindrical coordinates the cylinder always lines up along the z axis; use the **coordsys** command before issuing the **rm** command if the points to be removed are not aligned with the z-axis; then issue a final **coordsys** command to return to normal. Also note that the points that are removed become dudded out (point type set to 21) and are not removed from the data array.

The other options are:

`geometry -- xyz, rtz, rtp`

`ifirst, ilast, istride -- point range to search`

`xmin, ymin, zmin -- minimums of geometry type coordinates`

`xmax, ymax, zmax -- maximums of geometry type coordinates`

`xcen, ycen, zcen -- center of removal space for geometry`

`xscale, yscale, zscale -- scaling factors for geometry limits`

FORMAT:

rm /geometry/ifirst,ilast,istride/xmin,ymin,zmin/xmax,ymax,zmax/
xcen,ycen,zcen/[xscale,yscale,zscale]

EXAMPLE:

rm/xyz/0,0,0/2.,2.,2./4.,4.,4./0.,0.,0./

rm/rtz/0,0,0/0.,0.,0./1.,360.,10./0.,0.,0./

RMMAT

Removes all points of a specified material number. This command duds out the points (sets `ipt=21`) but doesn't remove them from the data array. Use **edit/parts** to find the correct material numbers. To actually remove the points see the **rmpoints** command.

FORMAT:

rmmat / material number/

RMPOINT

Removes a specified list of points (`ifirst, ilast, istride`) from a point distribution. The first format sets the point type flag [`itp=ifitpdud (21)`] to indicate that the set of points should be removed, but does not actually remove the points. The second format, **compress**, compresses and material-wise resequences all appropriately flagged points. If `iflag` is inclusive, any element containing a removed point is removed. IF `iflag` is exclusive (default), any element containing a retained point is retained.

FORMAT:

rmpoint/ifirst, ilast, istride/ iflag

rmpoint/compress/

rmpoint/zero_volume/threshold (Elements whose volumes are less than or equal to threshold will be removed.)

RMREGION

Removes points that lie within the specified region.

FORMAT:

rmregion/region_name/

RMSPHERE

Removes a sphere of points from a point distribution.

FORMAT:

rmsphere/inner_radius/outer_radius/xcen, ycen, zcen/

RMSURF

Removes points that lie in, on or in and on the specified surface. `ioper` can be one of the following:

- lt** - only points in the surface are removed
- eq** - only points on the surface are removed
- le** - all points in or on the surface are removed

FORMAT:

rmsurf/region_name/ioper

ROTATELN

Rotates a point distribution (specified by `ifirst, ilast, istride`) about a line. The **copy** option allows the user to make a copy of the original points as well as the rotated points, while **nocopy** just keeps the rotated points themselves. The line of rotation defined by `x1` through `z2` needs to be defined such that the endpoints extend beyond the point distribution being rotated. `theta` (in degrees) is the angle of rotation whose positive direction is determined by the right-hand-rule, that is, if the thumb of your right hand points in the direction of the line (1 to 2), then your fingers will curl in the direction of rotation. `xcen, ycen, zcen` is the point where the line can be shifted to before rotation takes place.

FORMAT:

rotateIn /ifirst, ilast, istride/ [no] copy / x1, y1, z1/x2, y2, z2/theta/
xcen, ycen, zcen/

ROTATEPT

Rotates a point distribution (defined by `ifirst, ilast, istride`) about a point `xcen, ycen, zcen`. `phi` (in degrees) is the angle of rotation of the XY plane around the Z-axis, where positive `phi` is measured from the positive x-axis toward the positive y-axis. `theta` (in degrees) is the angle of rotation toward the negative z-axis. The **(no) copy** options are as described in the **rotateIn** command.

FORMAT:

rotatept /ifirst, ilast, istride/ [no] copy / xcen, ycen, zcen/theta/phi

RZ

This command adds points to the mesh. It can distribute points evenly or according to a ratio zoning method.

xyz specifies Cartesian coordinates.

rtz specifies cylindrical coordinates.

rtp specifies spherical coordinates.

When using the **rtz** or **rtp** coordinate systems the center is at $(0, 0, 0)$. Use a **trans** command to move the center. For the **rtz** command, minimum and maximum coordinates are the triplets: radius from the cylinder's axis, angle in the xy-plane measured from the x-axis and height along the z-axis. For the **rtp** command minimum and maximum coordinates are the triplets: radius from the center of the sphere axis, angle in the zy-plane measured from the positive z-axis and the angle in the xy-plane measured from the positive x-axis (see III.a.11). Note that the **rtz** always results in a (partial) cylinder of points centered around the z axis. Use the **rotateln** command to orient the cylinder. For example, to center the cylinder around the y axis, specify the x axis as the line of rotation in the **rotateln** command.

ni,nj,nk number of points to be created in each direction.

xmin,ymin,zmin minimums for coordinates.

xmax,ymax,zmax maximums for coordinates.

iiz,ijz,ikz if =0 then mins and maxs are used as cell centers
if =1 then mins and maxs are used as cell vertices

iirat,ijrat,ikrat ratio zoning switches (0=off,1=on)

xrz,yrz,zrz ratio zoning value - distance is multiplied by this value for each subsequent point.

FORMAT:

**rz/xyz|rtz|rtp/ni,nj,nk/xmin,ymin,zmin/xmax,ymax,zmax/
iiz,ijz,ikz/[iirat,ijrat,ikrat/xrz,yrz,zrz/]**

EXAMPLES:

rz/xyz/5,3,10/0.,2.,0./1.,6,2/1,1,1/

This results in a set of 150 points, five across from x=0. to x=5., 3 deep from y=2. to y=6. and 10 high from z=0. to z=2.

rz/rtz/4,6,11/0.,0.,0./3.,360.,10./1,0,1/

This results in 264 points arranged around the z- axis. There are 3 rings of points at distances $r=1.$, $r=2.$ and $r=3.$ from the z-axis. There are 11 sets of these three rings of points and heights $z=0.$, $z=1.$, $z=2.$,..., $z=10.$ In each ring there are 6 points where each

pair of points is separated by 60°; note that $ijz=0$ requests that points be placed at cell centers, hence the first point will be at 30° not at 0°. There will be 6 points identical points at 11 intervals along the z-axis at heights $z=0.$, $z=1.$, $z=2.$,... $z=10$. **Filter** should be used to remove these duplicate points.

RZBRICK

Builds a brick mesh and generates a nearest neighbor connectivity matrix. This command is similar to the **rz** command format except here we have symmetry flags to input. A second format specifies that a mesh be created and connected.

xyz specifies Cartesian coordinates.

rtz specifies cylindrical coordinates.

rtp specifies spherical coordinates.

ni, nj, nk number of points to be created in each direction.

$xmin, ymin, zmin$ minimums for coordinates.

$xmax, ymax, zmax$ maximums for coordinates.

iiz, ijz, ikz if =0 then mins and maxs are used as cell centers
if =1 then mins and maxs are used as cell vertices

$iirat, ijratt, ikratt$ ratio zoning switches (0=off,1=on)

xrz, yrz, zrz ratio zoning value - distance is multiplied by the value for each subsequent point.

name name of pstatus containing starting point number

$isym, jsym, ksym$

FORMAT:

rzbrick/xyz|rtz|rtp $ni, nj, nk/xmin, ymin, zmin/xmax, ymax, zmax/$

$iiz, ijz, ikz/[iirat, ijratt, ikratt/xrz, yrz, zrz/isym, jsym, ksym]$

or

rzbrick/xyz|rtz|rtp $ni, nj, nk/pstatus, get, name/ connect/$

Use this option with QUADXYZ to connect logically rectangular grids.

RZS

Builds a sphere by generating coordinates of points and also modifies zoning by ratio-zoning point distributions. See the **rz** command for more details. The *itype* flag defines what type of sphere will be generated.

itype=1 generates a sphere by gridding the faces of a cube and then projecting the vertices onto a sphere.

itype=2 generates a sphere by subdividing an icosahedron placed on the surface of a sphere. itype= 1 or 2 distributes points only, call search to generate connectivity information.

nr is the number of radii

npt is the number of points total in the sphere

itype=8 generates a hexahedral icosahedron grid. This option distributes points and generated the grid connectivity data structures.

xirad,xorad are the inner and outer radii of the sphere. For itype=8 reverse inner and outer radii.

xcen,ycen,zcen are the coordinates of the center of the sphere

iz if =0 then mins and maxs are used as cell centers

if =1 then mins and maxs are used as cell vertices

irat is ratio zoning switch (0=off,1=on)

rz is ratio zoning value - distance is multiplied by the value for each subsequent point.

FORMAT:

rzs/itype/nr,npt,xirad,xorad/xcen,ycen,zcen/iz/irat,rz/

FORMAT:

rrs/8/5/162/1.0,0.5/0.,0.,0./1,0,0.0/

rrs/2/5/162/0.5,1.0/0.,0.,0./1,0,0.0/

SCALE

Scale a point distribution specified by ifirst,ilast,istride according to the scale factors iscale, jscale, and kscale. The letters i, j, and k in the scale factors correspond to coordinates specified by one of the geometry types [**xyz** (Cartesian), **rtz** (cylindrical), **rtp** (spherical)]. For example, if geometry = rtp then iscale = rscale, jscale = tscale, and kscale = zscale. If the scaling option is **relative** then the scaling factors are unitless multipliers with reference to some geometric center (xcen,ycen,zcen). If the scaling option is **absolute** then the scaling factors are consistent units added on to the existing coordinates

FORMAT:

scale/ifirst,ilast,istride/**absolute|relative/xyz|rtz|rtp**/
iscale,jscale,kscale/xcen,ycen,zcen

SEARCH

This is the main command for generating the connectivity list.

isrchopt -

0 => Set up the mesh for specified points. If points are not specified, set up the mesh for the entire problem. Also, remove the enclosing tetrahedron after generating the mesh.

1 => Same as 0 except do not remove tetrahedra associated with the enclosing tetrahedron.

2 => Add specified points to the existing mesh and remove tetrahedra associated with the enclosing tetrahedron.

3 => Add specified points to the existing mesh and do not remove tetrahedra associated with the enclosing tetrahedron.

4 => Just remove tetrahedra associated with the enclosing tetrahedron.

FORMAT:

search/isrchopt/ifirst,ilast,istride/

SETPTS

Sets point types and material regions by calling **surfset** and **regset** routines. Generate constraint table.

FORMAT:

setpts

SETTETS

Set tetrahedra color (material type). Mark interface points; create child points at interior boundaries. Points on interior

If there are no parameters **settets** sets the color of all tets based on previous mregion specification

If there are parameters, tets whose face centroids all lie within the box specified by points 1 and 2 are colored to **color**. **Color** often represents material type.

FORMAT:

settets

settets/color/x1,y1,z1/x2,y2,z2/

SMOOTH

The SMOOTH Command smoothes 2D or 3D mesh objects. Adaptive smoothing (to values of specified fields) or non-adaptive smoothing is available. In the first form, we adapt the current mesh object to the specified field of the reference mesh object (`cmo_ref`). Although the x-y-z values of `-cmo-` are altered by adaption, `cmo_ref` should never change. Hence, to accomplish adaption using one or more fields in the current mesh object itself, one should let `cmo_ref` be a copy of the current mesh object. The user can specify one of two algorithm choices: Minimum Error Gradient Adaption (**mega**), or Elliptic Smoothing for Unstructured Grids (**esug**). The results of adaption of the grid to the field can be altered by using one or more **field** commands beforehand to modify the field of `cmo_ref`. For example, by increasing the scale of a field using **field/scale**, the **esug** algorithm option of **smooth** will produce grids with increased numbers of nodes in the regions where the field experiences relatively large gradients. By volume averaging a field using **field/volavg**, **smooth** will cause a more gentle form of adaption with a better grading of elements. By composing the values of the field with **log** or **asinh** using **field/compose**, one can cause **smooth** to shift nodes to where the logarithm (or hyperbolic arcsine) of the field has interesting features, rather than where the field itself has interesting features. In the second form of adaptive smoothing the user supplies a subroutine call **fadpt**. In this case the seventh argument to the **smooth** command is the keyword **user**.

Subroutine fadpt(x,y,z,nvec,time,f)

PURPOSE-

Adaption function for smoothing algorithms. (Replace this code with a user-supplied function for nontrivial smoothing.) This default function should create a uniform grid when used with **mega** type smoothing.

INPUT ARGUMENTS -

`x,y,z` - Input spatial coordinate arrays.

`nvec` - Length of spatial arrays.

`time` - Current time (for time dependent adaption).

OUTPUT ARGUMENTS -

`f` - Array of adaption function values.

In the third form of the **smooth** command, we perform non-adaptive smoothing on the specified point set, using either **mega** or **esug**. You can specify an optional `control` value between zero and one. The default (`control=0.`) results in the standard smoothing scheme. Increasing `control` towards 1. causes the scheme to be progressively more controlled (moving the mesh less), until at `control=1`, there is no mesh movement whatsoever. By default, the second argument is `POSITION`. This results in the positions of the nodes being changed. **set,add,sub** are reserved for future implementation of smoothing using node velocities.

FORMAT:

```
smooth/position|set|add|sub /mega|esug/ifirst,ilast,istride/cmo_ref/field/
smooth/position|set|add|sub/mega|esug/ifirst,ilast,istride/user/
smooth/position|set|add|sub/mega|esug/ifirst,ilast,istride/control/
```

EXAMPLES:

1. Smooth the positions of all the nodes in the mesh (Here, missing arguments are supplied default values.)

```
SMOOTH
```

2. Smooth all nodes in the mesh, using controlled smoothing with `control = 0.5`

```
SMOOTH///1,0,0/0.5
```

3. Adaptively smooth interior nodes in the mesh using user-supplied FADPT subroutine.

```
PSET/interior/ZQ/ITP/1,0,0/0
SMOOTH///PSET,GET,interior/USER
```

4. Copy 2dmesh to 2dmesh_ref.

Normalize density field to have values of order unity. Adapt 2dmesh to normalized density field values in 2dmesh_ref.

```
CMO/COPY/2dmesh_ref/2dmesh
FIELD/SCALE/NORMALIZE/1.0/1,0,0/density
CMO/SELECT/2dmesh
SMOOTH///1,0,0/2dmesh_ref/density
```

SURFACE

Defines a boundary surface of the type specified in `ibtype`.

`ibtype` can be **free**, **interface**, **reflect**, **intrcons** or **virtual**. Use **reflect** or **free** for external boundaries, **interface** for interior interfaces, **intrcons** for constrained interior interfaces.

Use **virtual** for virtual interfaces.

The surface is defined by `istype` and `x1` through `z4`.

`istype` can be **plane**, **box**, **parallel**(piped), **sphere**, **cylinder**, **cone**, **ellipse**(oid), **tabular** (rotated tabular profile), or **sheet**.

`x1` through `z4` are specified with the surface type in mind.

`isurname` is the name of the surface and must be unique for each surface defined by **surface**.

FORMAT:

surface/`isurname`/`ibtype`/`istype`/`x1`/`y1`/`z1`/`x2`/`y2`/`z2`/`x3`/`y3`/`z3`/`x4`/`y4`/`z4`/
surface/`isurname`/`ibtype`/**sheet**/`cmo_name`/

EXAMPLES:

surface/`isurname`/`ibtype`/**box**/`xmin`,`ymin`,`zmin`/`xmax`,`ymax`,`zmax`/

surface/`isurname`/`ibtype`/**cone**/`x1`,`y1`,`z1`/`x2`,`y2`,`z2`/`radius`/

Where point 1 is the vertex and point 2 is the top center of the cone with radius from that point. A cone is finite but open. To create a closed cone cap the open end with a plane.

surface/`isurname`/`ibtype`/**cylinder**/`x1`,`y1`,`z1`/`x2`,`y2`,`z2`/`radius`/

Where point 1 is the bottom center and point 2 is the top center of the cylinder. Cylinders are open but finite. To create a closed cylinder cap both ends with planes.

surface/`isurname`/`ibtype`/**ellipse**/`x1`,`y1`,`z1`/`x2`,`y2`,`z2`/`x3`,`y3`,`z3`/`ar`,`br`,`cr`/

Where point 1 is the center of the ellipsoid and point 2 is on the a semi-axis (new x), point 3 is on the b semi-axis (new y), and `ar`, `br`, `cr` are radii on their respective semi-axes.

surface/`isurname`/`ibtype`/**parallel**/`x1`,`y1`,`z1`/`x2`,`y2`,`z2`/`x3`,`y3`,`z3`/`x4`,`y4`,`z4`/

Where points 1, 2, 3 are the front left, front right and back left points of the base and point 4 is the upper left point of the front face.

surface/`isurname`/`ibtype`/**plane**/`x1`,`y1`,`z1`/`x2`,`y2`,`z2`/`x3`,`y3`,`z3`

surface/`isurname`/`ibtype`/**planexyz**/`x1`,`y1`,`z1`/`x2`,`y2`,`z2`/`x3`,`y3`,`z3`

the direction of the normal to the plane is determined by the order of the points according to the right hand rule.

surface/`isurname`/`ibtype`/**planertz**/`radius1`,`theta1`,`z1`,

`radius2`,`theta2`,`z2`,`radius3`,`theta3`,`z3`,`xcen`,`ycen`, `zcen`/

surface/isurname/ibtype/**planertp**/radius1,theta1,phi1,
radius2,theta2,phi2, radius3,theta3,phi3, xcen,ycen,zcen/
surface/isurname/ibtype/**sheet**/cmo_name/

Sheet surfaces may be input by specifying a cmo_name. The Mesh Object must be either a 2D quad Mesh Object or a 2D triangle Mesh Object.

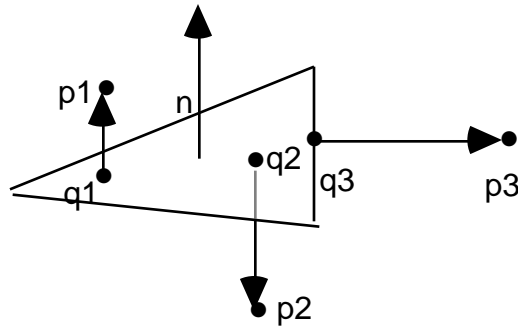
Inside/outside with respect to sheet surfaces will be determined by the following algorithm:

- For the point being considered, p, find the nearest sheet triangle and the closest point, q, to p that lies on that triangle.
- Construct the vector, \vec{d} , from q to p.

- Construct the outward normal to the triangle, \vec{n} . The outward normal is constructed using the right hand rule and the order of the points in the sheet. Sheets may be specified as quad Mesh Object (i.e. a 2 dimensional array of points containing the coordinates of the corners of each quad). Either two triangles (divide each quad in two using point (i,j) and (i+1,j+1)) or four triangles (add a point in the center of the quad) are generated by each quad. Applying the right hand rule to the points (i,j), (i+1,j), (i+1,j+1) gives the direction of the normal for all triangles created from the quad.

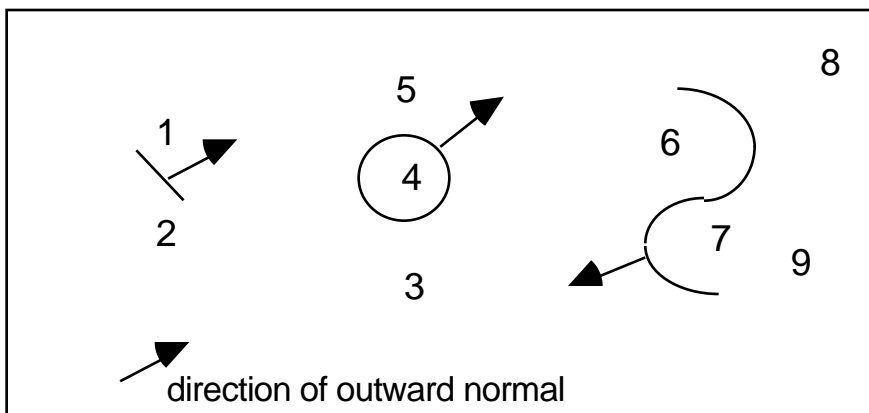
- If $\vec{d} \cdot \vec{n} < 0$ then the point is inside. If $\vec{d} \cdot \vec{n} > 0$ the point is outside. If $\vec{d} \cdot \vec{n} = 0$, and if p is on the triangle then p=q and p is on the triangle.

- If $\vec{d} \cdot \vec{n} = 0$ and p is not on the triangle then p is outside.



p1 is outside
p2 is inside
p3 is outside

One implication of this definition is that the concept of shadows cast by open sheets no longer is valid. Sheets may be considered to extend to the boundary of the geometry.



points 1, 5, 3, 6 are outside
points 2, 4, 7, 8 and 9 are inside

surface/isurname/ibtype/**sphere**/xcen,ycen,zcen/radius/
surface/isurname/ibtype/**tabular**/x1,y1,z1/x2,y2,z2/geom/
r1,z1

```

r2,z2
r3,z3
....
rn,zn

end
or
r1,theta1
r2,theta2
r3,theta3
...
rn,thetan
end

```

Where point 1 and point 2 define the axis of rotation for the tabular profile with point 1 as the origin. This is followed by pairs of profile descriptors depending on the value of `geom`. If `geom` is set to **rz**, then the `r` value is a radius normal to the axis of rotation and `z` is the distance along the new axis of rotation. If `geom` is set to **rt** then `theta` is the angle from the axis of rotation at point 1 and `r` is the distance from point 1 along `theta`. The first pair must start on a new line and all lines must contain pairs of data. The last pair of data must be followed by `end`.

SURFPTS

Generates points on boundary surfaces previously defined by the `surface` or `region` command. The variable `itype` can be **surface** or **region** and `iname` is the name of the surface or region. The points are generated by shooting rays through a user specified set of points from an origin point, line or plane and finding the surface intersection of each ray. The point location for a region is determined by `iregpt` and can be on the **inside**, **outside** or **both** surfaces.

FORMAT:

```

surfpts/itype/iname/iregpt/ifirst,ilast,istride/geom/ray_origin
surfpts/itype/iname/iregpt/pset,get,setname/geom/ray_origin

```

Where `ifirst,ilast,istride` or **pset,get,setname** define the set of points to shoot rays through.

SPECIFICALLY FOR ALLOWABLE GEOMETRY TYPES:

surfpts/itype/iname/iregpt/ifirst,ilast,istride/

xyz/x1,y1,z1/x2,y2,z2/x3,y3,z3/

Where points 1, 2, 3 define the plane to shoot rays from that are normal to the plane.

surfpts/itype/iname/iregpt/ifirst,ilast,istride/ **rtz**/x1,y1,z1/x2,y2,z2/

Where points 1, 2, define the line to shoot rays from that are perpendicular to the line.

surfpts/itype/iname/iregpt/ifirst,ilast,istride/ **rtp**/xcen,ycen,zcen/

surfpts/itype/iname/iregpt/ifirst,ilast,istride/ **points**

/iffirst,iflast,ifstride/

Where ifirst,ilast,istride define a set of points to shoot rays from.

TRANS

Translates a selected set of points (ifirst,ilast,istride) in X,Y,Z space by picking one specific point (xold,yold,zold) in the set of points and moving it to new coordinates (xnew,ynew,znew) with a linear translation. This will then cause the remaining points in the set to be moved by the same translation.

FORMAT:

trans/ifirst,ilast,istride/xold,yold,zold/xnew,ynew,znew

EXAMPLE:

trans/pset,get,mypoints/0.,0.,0./2.0,2.0,0./

The points in the **pset** mypoints will be moved 2 in the positive x direction and 2 in the positive y direction.

ZQ

Set or print node attribute values of a selected set of nodes.

To print, omit the 'value' field.

For printing, attributes are grouped as follows:

Group1: isq,imt,itp (material type and point types)

Group2: x,y,z (coordinates)

To print, specify any one of a group and all will be printed.

To set an attribute value, set value and all selected nodes will be set to this value.

Attribute added with a **cmo/addatt** command may also be printed or changed.

FORMAT:

zq/iattribute/ifirst,ilast,istride/value

EXAMPLE:

zq/imt/1,100,2/material1/

will set all odd numbered points between 1 and 100 to material type 'material 1'

zq/xic/1,0,0/

will print coordinates of all points

IV. Interfacing User Routines to X3D

a. *Building an executable and running X3D.*

The executable is built by linking a driver routine with the code and utility libraries. The driver routine must contain a call to `initx3d` and a call to `dotaskx3d` and must contain a subroutine called `user_sub`. A sample driver routine is listed:

```
      program adrivgen
      C
      C
      #####
      C
      C      PURPOSE -X3D driver
      C
      #####
      C
      C      implicit real*8 (a-h,o-z)
      C
      C      call initx3d('generate','noisy',' ',' ')
      C
      C      call dotaskx3d('interact',ierror_return)
      C
      C      stop
      C      end
      C
      C      subroutine user_sub(imsgin,xmsgin,cmssgin,msgtyp,nwds,
      x                          ierr1)
      C
      #####
      C
      C      PURPOSE -
      C
      C      Process user supplied commands
      C
      C      INPUT ARGUMENTS -
      C
      C      imsgin - integer array of tokens returned by parser
      C      xmsgin - real array of tokens returned by parser
      C      cmssgin - character array of tokens returned by parser
      C      msgtyp - int array of token types returned by parser
      C      nwds - number of tokens returned by parser
      C
      C      OUTPUT ARGUMENTS -
      C
      C      ierr1 - 0 for successful completion - -1 otherwise
      C
      #####
      C      character*32 cmssgin(nwds)
      C      integer imsgin(nwds),msgtyp(nwds)
```

```

        integer nwds,ierr1,lenc
        real*8 xmsgin(nwds)
C   get command length
        lenc=icharlnf(cmsgin(1))
C   set default error return to fail
        ierr1=-1
C   Insert code here to handle user coded subroutines
C   For example
C       if(cmsgin(1)(1:lenc).eq.'my_cmnd')      then
C           call my_rtn(imsgin,xmsgin
C       *               cmsgin,msgtyp,nwds,ierr1)
C       else
C           ierr1=-1
C       endif
C
        return
        end

```

Sample build scripts for the supported platform are:

Sun OS and Sun Solaris

```
f77 -g -o x3dgen adrivgen.f libx3d.a libutil.a
```

IBM RISC

```
f77 -g -o x3dgen -qintlog -brenam:..fdate,.fdate_ adrivgen.f
libx3d.a libutil.a
```

SGI

```
f77 -g -Nn10000 -o x3dgen adrivgen.f libx3d.a libutil.a
```

HP

```
f77 -g +U77 -R8 -o x3dgen adrivgen.f libx3d.a libutil.a
```

Once the executable is built, the dictionary file must be installed. This file, `x3ddict`, is supplied with the libraries. It must either exist in the directory from which X3D will be run, or an environment variable may be set to give the directory path to its location.

The format of the `setenv` command is:

```
setenv x3ddict full_directory_path_to_x3ddict.
```

To execute, use standard unix file redirection for standard input and output. X3D will produce two additional files, `outx3dgen` and `logx3dgen`. These contain detailed output information and the list of commands respectively. X3D may also be run interactively in which case the user will be prompted to enter commands from the workstation.

b. Issuing Commands from a user program.

Any X3D command can be issued by calling the subroutine `dotaskx3d`, for example:

```
call dotaskx3d('cmo/select/3dmesh', ierr1)
```

will select the Mesh Object named *3dmesh*. *ier1* will be zero if the commands are executed with no error, non-zero otherwise.

By using the X3D command **infile**, a series of commands may be executed, for example

```
call dotaskx3d('infile/mydeck', ier1)
```

will execute all the X3D commands that are in the user's file named *mydeck*. The final command in the file *mydeck* should be **finish**.

c. **Writing user commands**

The access to user written subroutines is through the X3D subroutine, *user_sub*. It is passed the parsed command input line. The parser breaks up the input line into tokens and returns to X3D a count of number of tokens, an array containing the token types, and the tokens themselves. The parameters returned by the parser are:

```
nwds ( number of tokens)
msgtyp (integer array of token types - 1 for integer, 2 for real, 3 for
        character, msgtyp(nwds+1) = -1)
imsgin (array of integer tokens, e.g. imsgin(i) is the ith token which is an
        integer if msgtyp(i)=1)
xmsgin (array of real tokens)
cmsgin (array of character tokens)
```

Null fields are given the integer value 0, real value 0. and character value '-def-'. The parser is written in C, therefore character variables returned will be null terminated on some platforms. A FORTRAN function is supplied; *icharlnf* will return the length of the character string blank or null terminated, ignoring leading blanks

If the user has written a subroutine, *my_routine*, that responds to the command, *my_comnd*, the call from *user_sub* should look like:

```
elseif (cmsgin(1)(1:lenc).eq. 'my_comnd')
x      call my_routine(nwds,imsgin,xmsgin,cmsgin,msgtyp,ierr1)
```

The subroutine *my_routine* should set *ierr1* to zero if the command is processed successfully and should use the *cmo* interface routines to access the components of the Mesh Object that it needs, for example:

```
character*32 cmo
pointer (ipimt1, imt1(*))
c      get the name of the current mesh object
call cmo_get_name(cmo_name,ierror)
c      get the number of nodes and the material ids
call cmo_get_info('nnodes',cmo_name,nnodes,ilen,ityp,ierr)
call cmo_get_info('imt1',cmo_name,ipimt1,ilen,ityp,ierr)
```

The subroutine `user_sub` is supplied with the driver and is defaulted to print the error message: 'Illegal command' and return.

- d. The following template is an example of using the an existing mesh object and of creating a new mesh object. The existing mesh object is a 3d object. The object to be created is a 2d object. It is first necessary to set up the pointer statements for both the existing and new mesh objects.

```
C  Definitions for incoming (existing) cmo
C
    pointer (ipimt1, imt1)
    pointer (ipitp1, itp1)
    pointer (ipicr1, icr1)
    pointer (ipisn1, isn1)
    integer imt1(1000000), itp1(1000000),
*         icr1(1000000), isn1(1000000)
    pointer (ipxic, xic)
    pointer (ipyic, yic)
    pointer (ipzic, zic)
    dimension xic(1000000), yic(1000000), zic(1000000)
    pointer (ipitetclr, itetclr)
    pointer (ipitettyp, itettyp)
    pointer (ipitetoff, itetoff)
    pointer (ipjtetoff, jtetoff)
    pointer (ipitet, itet)
    pointer (ipjtet, jtet)
    integer itetclr(1000000), itettyp(1000000),
*         itetoff(1000000), jtetoff(1000000)
    integer itet(4,1000000) , jtet(4,1000000)

C
C  Definitions for cmo that is to be created
C
    pointer (ipimtla, imtla)
    pointer (ipitpla, itpla)
    pointer (ipicr1a, icr1a)
    pointer (ipisn1a, isn1a)
    integer imtla(1000000), itpla(1000000),
*         icr1a(1000000), isn1a(1000000)
    pointer (ipxica, xica)
    pointer (ipyica, yica)
    pointer (ipzica, zica)
    dimension xica(1000000), yica(1000000), zica(1000000)
    pointer (ipitetclra, itetclra)
    pointer (ipitettypa, itettypa)
    pointer (ipitetoffa, itetoffa)
    pointer (ipjtetoffa, jtetoffa)
    pointer (ipiteta, iteta)
    pointer (ipjteta, jteta)
    integer itetclra(1000000), itettypa(1000000),
*         itetoffa(1000000), jtetoffa(1000000)
```

```

integer iteta(3,1000000) , jteta(3,1000000)

C   Get the existing cmo - its name is in the variable cmoin
C
call cmo_get_name(cmoin,ier)
C
C   Get the scalar mesh variables
call cmo_get_info('nnodes',cmoin,npoints,lencm,itypcm,ier)
call cmo_get_info('nelements',cmoin,ntets,lencm,itypcm,ier)
call cmo_get_info('ndimensions_topo',cmoin,ndt,lencm,itypcm,ier)
call cmo_get_info('ndimensions_geom',cmoin,ndg,lencm,itypcm,ier)
call cmo_get_info('nodes_per_element',cmoin,npe,lencm,itypcm,ier)
call cmo_get_info('faces_per_element',cmoin,nfpe,lencm,itypcm,ier)
call cmo_get_info('mbndry',cmoin,mbndry,lencm,itypcm,ier)
C
C   Get pointers to the vector variables
call cmo_get_info('ialias',cmoin,ipialias,lenialias,ictype,ier)
call cmo_get_info('imtl',cmoin,ipimtl,lenimtl,ictype,ier)
call cmo_get_info('itpl',cmoin,ipitpl,lenitpl,ictype,ier)
call cmo_get_info('icrl',cmoin,ipicrl,lenicrl,ictype,ier)
call cmo_get_info('isnl',cmoin,ipisnl,lenisnl,ictype,ier)
call cmo_get_info('xic',cmoin,ipxic,lenxic,ictype,ier)
call cmo_get_info('yic',cmoin,ipyic,lenyic,ictype,ier)
call cmo_get_info('zic',cmoin,ipzic,lenzic,ictype,ier)
call cmo_get_info('itetclr',cmoin,ipitetclr,lenitetclr,ictype,ier)
call cmo_get_info('itettyp',cmoin,ipitettyp,lenitettyp,ictype,ier)
call cmo_get_info('itetoff',cmoin,ipitetoff,lenitetoff,ictype,ier)
call cmo_get_info('jtetoff',cmoin,ipjtetoff,lenjtetoff,ictype,ier)
call cmo_get_info('itet',cmoin,ipitet,lenitet,ictype,ier)
call cmo_get_info('jtet',cmoin,ipjtet,lenjtet,icmotype,ier)
C
C   Create the new 2d cmo - call it cmoout.
C
call cmo_exist(cmoout,ier)
C
C   ier.eq.0 means that the cmo already exists - if so release it.
C
if(ier.eq.0) call cmo_release(cmoout,idelete)
C
C   Set active cmo to cmoout
call cmo_set_name(cmoout,ier)
C
C   set scalar mesh variables
C
call cmo_set_info('nnodes',cmoout,npoints,1,1,ier)
call cmo_set_info('nelements',cmoout,ntets,1,1,ier)
C
C   the following scalars need to be set for a 2d cmo
C
call cmo_set_info('ndimensions_topo',cmoout,2,1,1,ier)
call cmo_set_info('ndimensions_geom',cmoout,3,1,1,ier)
call cmo_set_info('nodes_per_element',cmoout,3,1,1,ier)
call cmo_set_info('faces_per_element',cmoout,3,1,1,ier)

```

```

C
C   allocate memory for vector variables
C   call cmo_newlen(cmoout,ier)
C
C   now get the pointers to the allocated memory for the vector data
C   call cmo_get_info('imt1',cmoout,ipimtla,lenimtla,icmotype,ier)
C   call cmo_get_info('itp1',cmoout,ipitpla,lenitpla,icmotype,ier)
C   call cmo_get_info('icr1',cmoout,ipicr1a,lenicr1a,icmotype,ier)
C   call cmo_get_info('isn1',cmoout,ipisn1a,lenisn1a,icmotype,ier)
C   call cmo_get_info('xic',cmoout,ipxica,lenxica,icmotype,ier)
C   call cmo_get_info('yic',cmoout,ipyica,lenyica,icmotype,ier)
C   call cmo_get_info('zic',cmoout,ipzica,lenzica,icmotype,ier)
C   call cmo_get_info('itetclr',cmoout,ipitetclra,lenclra,icmotype,ier)
C   call cmo_get_info('itettyp',cmoout,ipitettypa,lentypa,icmotype,ier)
C   call cmo_get_info('itetoff',cmoout,ipitetoffa,lenoffa,icmotype,ier)
C   call cmo_get_info('jtetoff',cmoout,ipjtetoffa,lenoffa,icmotype,ier)
C   call cmo_get_info('itet',cmoout,ipiteta,leniteta,icmotype,ier)
C   call cmo_get_info('jtet',cmoout,ipjteta,lenjteta,icmotype,ier)
C
C   now the values for the vector components of the 2d mesh
C   object can be set.

```

e. Utility subroutines

The following subroutines are available to code developers who wish to add modules to X3D. In the subroutine definitions that follow, input arguments are underlined.

1. Memory Manager

X3D uses dynamic memory allocation. Memory is referenced by a two part name, block name and partition name. It is allocated in integer or real blocks. Each memory block is preceeded by a header and terminated by a trailer. The memory manager always returns the pointer to the data section of the memory block. Length is specified in words. Type indicates if the words are integer or real. Different platforms will have different values for integer and real word lengths. These machine dependent values are collected in the include file `machine.h`.

Allocate a block of memory:

mmgetblk(blkin,prtin,iadr,length,itype,icscode)

blk	in	block name of memory block
prt	in	partition name of memory block
iadr		pointer to memory block (data section)
length		number of words to be allocated
itype		1 for integer, 2 for real
icscode		return code, 0 for no errors

Release a block of memory:

mmrelblk(blkin,prtin,iadr,icscode)

iadr	is not used
------	-------------

Release a partition of memory -- all blocks belonging to this partition will be released:

mmrelprt(prtin,icscode)

Increment a block of memory:

mmincblk(blkin,prtin,iadr,increment,icscode)

increment	number of words to increment memory block
-----------	---

Find pointer to a block of memory:

mmfindbk(blkin,prtin,iadr,length,icscode)

iadr	pointer to memory block (data)
length	number of words allocated

Return type of a block of memory:

mmgettyp(ipin,itypout,icscode)

ipin	pointer to memory block (data)
itypout	type of data 1 for integer, 2 for real

Return number of words in a block of memory:

mmgetlen(ipin,lenout,icscode)

lenpout	number of words in a memory block
---------	-----------------------------------

Return name of a block of memory:

mmgetnam(ipin,blkout,prtout,icscode)

blkout	block name
prtout	partition name

Print a dump of allocated memory. This is useful for debugging purposes; the dump is listed in two parts, by time of allocation and by increasing pointer address:

mmprint()

Verify memory integrity. Print debug information if the memory block headers or trailers have been overwritten.

mmverify()

2. Mesh Object

cmo_create(cmo-name,ierror)

Create a new mesh object called *name*

cmo-name	name of new mesh object
----------	-------------------------

ierror	error return - 0 if no errors
--------	-------------------------------

cmo_get_info(ioption,cmo-name,iout,lout,itype,ierror)

Get values of scalar attribute of the mesh object. Get pointers to vector attributes

ioption	name of mesh object attribute whose value is to be retrieved; the information retrieved may be one of these key words or it may be the name of a user supplied attribute (generated by a cmo_addatt command):
---------	--

number_of_attributes

nnodes (number of nodes in the mesh)

nelements (number of elements in the mesh)

nfaces (number of unique topological facets)

nedges (number of unique edges in mesh) --

mbndry (boundary node flag value)

ndimensions_topo (topological dimensionality)

	ndimensions_geom nodes_per_element
	edges_per_element
	faces_per_element
	isetwd (pset membership information)
	ialias (alternate node numbers)
	imt1 (node material)
	itp1 (node type)
	icr1 (constraint numbers for nodes)
	isn1 (child, parent node correspondence)
	ign1 (igeneration numbers for nodes)
	xic, yic, zic (node coordinates)
	itetclr (integer array of element material)
	itettyp (geometry of element)
	xtetwd (eltset membership information)
	itetoff (index into itet array for an element)
	jtetoff (index into jtet array for an element)
	itet (node vertices for each element)
	jtet (element connectivity)
cmo-name	name of mesh object to be retrieved
iout	value of attribute if the attribute is a scalar or a pointer to the attribute if the attribute is a vector
lout	length of retrieved attribute
itype	type of attribute (1= integer, 2=real, 3=character)
ierror	return flag (0 if no errors)

cmo_set_info(ioption,cmo-name,data,lin,itype,ierror)

Set values of scalar attribute of the mesh object. Vector attributes are set by filling arrays pointed to by the vector attribute pointer

ioption	lattribute name
data	value of attribute of the salar attribute to be set
lin	length of attribute (1 for scalars)
itype	type of attribute (1= integer, 2=real, 3=character)

cmo_get_name(cmo-name,ierror)

Get the name of the current mesh object.

cmo-name	name of current mesh object
----------	-----------------------------

cmo_set_name(cmo-name,ierror)

Set the name of the current mesh object.

cmo_get_attribute_name(cmo-name,attribute-index,attribute-name,ierror)

This routine is useful when looping through all the attributes of a mesh object. To get the number of attributes use
cmo_get_info('number_of_attributes',...

attribute-index number of attribute

attribute-name name of retrieved attribute

cmo_newlen(cmo-name,ierror)

Adjust memory associated with mesh object. Must be called whenever the size of the mesh is adjusted in order to provide memory to the pointered attributes.

cmo_release(cmo-name,ierror)

Release a mesh object called cmo-name and release its memory

cmo-name name of mesh object

ierror error return - 0 if no errors

get_info_c(parameter-name,cmo-name,'sbcmaprm','default',cdata,ierror)

get_info_i(parameter-name,cmo-name,'sbcmaprm','default',idata,ierror)

get_info_r(parameter-name,cmo-name,'sbcmaprm','default',rdata,ierror)

Retrieve a mesh object parameter value: 'c' retrieve character data, 'i' integer data and 'r' real data.

parameter-name name of mesh object parameter

cmo-name name of mesh object

cdata,idata,rdata value of parameter

ierror error return - 0 if no errors

3. Point Selection

getptyp(point_type_name,point_type,ierror)

This routine converts point type names to point types.

See II.a for a list of point types, names and meanings

point_type_name name of point type

point_type value of point type

unpackpc(npoints,itp,isn,iparents)

This routine returns in the array iparents the parent point corresponding to each child point i, if point i is a child point. Ordinary points are their own parents.

npoints number of nodes

itp1	array of point types
isn1	array of parent child links
iparents	array of parent node number for each point that is a child point. - zero otherwise

unpacktp(ioptitp,iopt2,inum,ipitp1,ipitp2,ierror)

This routine sets, or's in, or and's in (depending on iopt2) a 1 in the array pointed to by ipit2 for each point that fits the criterion specified by ioptitp. A zero is set, or'd or and'd otherwise.

ioptitp	criterion
	allreal (0 itp1(i) 19)
	interior (itp1(i)=0)
	inteintf (itp1(i)=2,3,4)
	matlintr (itp1(i)=2,4,8,9,12,13,15,19)
	boundary (8 itp1(i) 19)
	reflect (itp1(i)=9,10, 12, 14, 15,16,18,19)
	free (itp1(i)=8,9,11, 13, 14, 15,17,18)
	intrface (itp1(i)=2,3,4,8,9,12, 13,15,16,17,18,19)
	virtual (itp(i)=3,4,8,9,16,17,18,19)
	removed (20 itp1(i) 29)
	merged (itp1(i)=20)
	dudded (itp1(i)=21)
iopt2	operation
	set set itp2 to 1 or 0
	or or in a 1 or 0 in itp2
	and and in a 1 or 0 in itp2
inum	number of nodes in itp1 array
ipitp1	pointer to array of point types
ipitp2	pointer to output array of 1's or 0's (length inum)

4. Character Length

Because X3D uses a parser written in C whereas most other modules are written in FORTRAN, the user must be very careful in using character comparison. Some character strings will be terminated with a blank (FORTRAN) and some by a null (C).

The following functions are provided to return character string length (number of characters in `iword` ignoring terminator character).

icharln(iword) Search for terminating blank or null.

icharlnf(iword) Ignore leading blanks then search for terminating blank or null.

icharlnb(iword) Search backwards for first blank or null - uses FORTRAN function **len** to give starting point (this is a risky assumption)

5. Retrieving Point Sets and Element Sets

eltlime returns an array of element numbers where the elements belong to the **eltset** given in the argument list. Eltsets must be specified by name. On return the array pointed to by `ipmpary` will contain the mpno element numbers that belong to the **eltset**.

eltlimc(ich1,ich2,ich3,ipmary,mpno,ntets,xtetwd)

<code>ich1,ich2,ich3</code>	eset,get,eltset_name
<code>ipmpary</code>	pointer to array of elements of <code>eltset_name</code>
<code>mpno</code>	number of elements in <code>eltset_name</code>
<code>ntets</code>	number of elements in mesh object
<code>xtetwd</code>	array of eltset membership information

pntlimc,pntlimn return an array of node numbers where the nodes belong to the **pset** given in the argument list. On return the array pointed to by `ipmpary` will contain mpno node numbers. These numbers are the nodes that belong to the **pset**.

pntlimc(ich1,ich2,ich3,ipmary,mpno,npoints,isetwd,ity1)

<code>ich1,ich2,ich3</code>	pset,get,pset_name
<code>ipmpary</code>	pointer to array of node number of <code>pset_name</code>
<code>mpno</code>	number of nodes in <code>pset_name</code>
<code>npoints</code>	number of nodes in mesh object
<code>isetwd</code>	array of pset membership information
<code>ity1</code>	array of point types

pntlimn(ich1,ich2,ich3,ipmary,mpno,npoints,isetwd,ity1)

<code>ity1,ity2,ity3</code>	first node, last node, stride
-----------------------------	-------------------------------

6. Array CompressionThe following utility routines compress arrays. Note that the output array may be the same as the input array in which case the compression is done in place. Also the mask array may be the same as the input array. The name suffixes of the compression routine may be decoded as **m** minus (negative), **n** non-zero, **p** positive, **z** equal to zero. If the routine name ends in **rrr**, the mask, input and

output arrays are all real. If the name ends in a single **r**, the mask is real, the input and output arrays are integers. Otherwise the mask, input and output arrays are all integers. For example `kmprsn(100,int,1,int,1,int,1,num)` will compress all the zeros out of array `int`.

kmprsm(n,z,iz,x,ix,y,iy,count)

<u>n</u>	length of z and x
<u>z</u>	array of masks
<u>iz</u>	stride in z
<u>x</u>	array of source
<u>ix</u>	stride in x
<u>y</u>	array of output
<u>iy</u>	stride in y
count	length of y

kmprsn(n,z,iz,x,ix,y,iy,count)

kmprsnr(n,z,iz,x,ix,y,iy,count)

kmprsnrrr(n,z,iz,x,ix,y,iy,count)

kmprsp(n,z,iz,x,ix,y,iy,count)

kmprspr(n,z,iz,x,ix,y,iy,count)

kmprsz(n,z,iz,x,ix,y,iy,count)

kmprszr(n,z,iz,x,ix,y,iy,count)

X3D REFERENCES:

GEOMETRY

Khamayseh, Ahmed; Ortega, Frank; Trease, Harold, "Ray Tracing for Point Distribution in Unstructured Grid Generation", LA-UR-95-4470.

Khamayseh, Ahmed; Ortega, Frank; Kuprat, Andrew, "A Robust Point Location algorithm for General Polyhedra", Journal of Computer Aided Geometric Design, LA-UR-95-4465

2-D VORONOI GRIDS:

Trease, H.E. (1981), "A Two-Dimensional Free Lagrangian Hydrodynamics Model," Ph.D. Thesis, University of Illinois, Urbana-Champaign.

3-D VORONOI GRIDS:

Trease, H.E. "Three-Dimensional Free Lagrangian Hydrodynamics," Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics, Springer-Verlag, Vol. 238, pp. 145-157, 1985.

3-D MEDIAN GRIDS (X3D):

Fraser D., "Tetrahedral Meshing Considerations for a Three-Dimensional Free-Lagrangian Code," Los Alamos National Laboratory report, LA-UR-88-3707, 1988.

Sahota, M.S., "Delaunay Tetrahedralization in a Three-Dimensional Free-Lagrangian Multimaterial Code," Proceedings of the Next Free-Lagrange Conference, Jackson Lake Lodge, Wyoming, June 3-7, 1990, Springer-Verlag Press, Vol. 395, pp. 130-138.

Ahmed Khamayseh, Andrew Kuprat, and Frank Ortega, "A Robust Point Location Algorithm for General Polyhedra," (to appear in Computer Aided Geometric Design)

3-D UNSTRUCTURED TETRAHEDRAL GRID RECONNECTION ALGORITHMS (X3D):

Trease, H.E. "Three-Dimensional Free Lagrangian Hydrodynamics," Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics, Springer-Verlag, Vol. 238, pp. 145-157, 1985.

Fraser D., "Tetrahedral Meshing Considerations for a Three-Dimensional Free-Lagrangian Code," Los Alamos National Laboratory report, LA-UR-88-3707, 1988.

Painter, J.W. and Marshall, J.C., "Three-Dimensional Reconnection and Fluxing Algorithms," Proceedings of the Next Free-Lagrange Conference, Jackson Lake Lodge, Wyoming, June 3-7, 1990, Springer-Verlag Press, Vol. 395, pp. 139-148.

Trease, H.E. "Parallel Nearest Neighbor Calculations," Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics, Springer-Verlag, Vol. 395, pp. 149-156, 1985.

DIFFUSION EQUATION COUPLING COEFFICIENT CALCULATIONS (X3D):

Trease, H.E. "Three-Dimensional Free Lagrangian Hydrodynamics," Proceedings of the first Free-Lagrange Conference, Lecture Notes in Physics, Springer-Verlag, Vol. 238, pp. 145-157, 1985.

Sahota, M.S., "An Explicit-Implicit Solution of the Hydrodynamic and Radiation Equations," Proceedings of the Next Free-Lagrange Conference, Jackson Lake Lodge, Wyoming, June 3-7, 1990, Springer-Verlag Press, Vol. 395, pp. 57-65.

Trease, H.E. and Dean, S.H., "Thermal Diffusion in the X-7 Three-Dimensional Code," Proceedings of the Next Free-Lagrange Conference, Jackson Lake Lodge, Wyoming, June 3-7, 1990, Springer-Verlag Press, Vol. 395, pp. 193-202.

UNSTRUCTURED GRID SMOOTHING ALGORITHMS:

Ahmed Khamayseh and Andrew Kuprat, "Anisotropic Smoothing and Solution Adaption for Unstructured Grids," LA-UR-95-2205, International Journal for Numerical Methods in Engineering, (submitted).

Kuprat, Andrew, "Adaptive Smoothing Techniques for 3-D Unstructured Meshes", LA-UR-96-1116.

Kuprat, Andrew, et al. "Moving Adaptive Unstructured 3-D Meshes in Semiconductor Process Modeling Applications", VLSI Journal, LA-UR-95-4128.

UNSTRUCTURED GRID ADAPTIVE MESH REFINEMENT (AMR):

Trease, H.E., "Adaptive Mesh Refinement (AMR) On Unstructured Tetrahedral Grids", to be published.